
Das 32000-
Sonderheft
Bausteine, Programmierung
Systeme, Anwendungen
Sonderheft Nr. 218
Preis DM 28.–
öS 210.–, sfr 28.–
Elektronik

Vorwort

„32000“ – diese Zahl ist nicht nur die Typenbezeichnung eines Mikroprozessors, sie steht vielmehr für eine Bausteinfamilie, deren Mitglieder aus CPUs, MMUs, FPUs und anderen Unterstützungs-Chips bestehen. Außerdem gehören dazu Software, Entwicklungswerkzeuge und Platinensysteme.

Was die 32000-Familie besonders interessant macht, ist das Konzept, das auf einer Philosophie basiert, die die bequeme Lösung von Anwenderproblemen zum Ziel hat. Wesentliche Merkmale sind beispielsweise eine Architektur, die für alle CPU-Versionen Kompatibilität garantiert, und eine Befehlsstruktur, die für problemlose Programmierung sorgt.

Das Know-how, das bei der Entwicklung der 32000-Familie in die Bausteine einfloß, hilft dem Anwender bei der Lösung komplexer Aufgaben. Moderne Halbleitertechnologie sorgt dafür, daß sich mit den Chips Systeme realisieren lassen, die trotz hoher Arbeitsgeschwindigkeit mit wenig Betriebsleistung auskommen und zuverlässig ihre Aufgaben erfüllen.

Im vorliegenden Heft sind alle derzeit verfügbaren Informationen über diese bemerkenswerte Bausteinfamilie zusammengetragen. Potentielle Anwender, aber auch Entwickler und Programmierer, die mit 32000-Chips arbeiten, finden hier eine Fülle von interessanten Fakten sowie nützlichen Hinweisen, die eine wertvolle Ergänzung der bisher verfügbaren Dokumentation, z. B. Datenbücher und Firmenschriften, darstellen.

Neben einigen grundlegenden Beiträgen, die in früheren Ausgaben der ELEKTRONIK erschienen sind, findet der Leser eine Fülle neuer Aufsätze, die alle Aspekte dieser Bausteinfamilie und ihrer Anwendung beleuchten. Mit diesem Sonderheft steht Entwicklern und Anwendern eine kompakte, aktuelle Informationsquelle zur Verfügung.

Die Redaktion

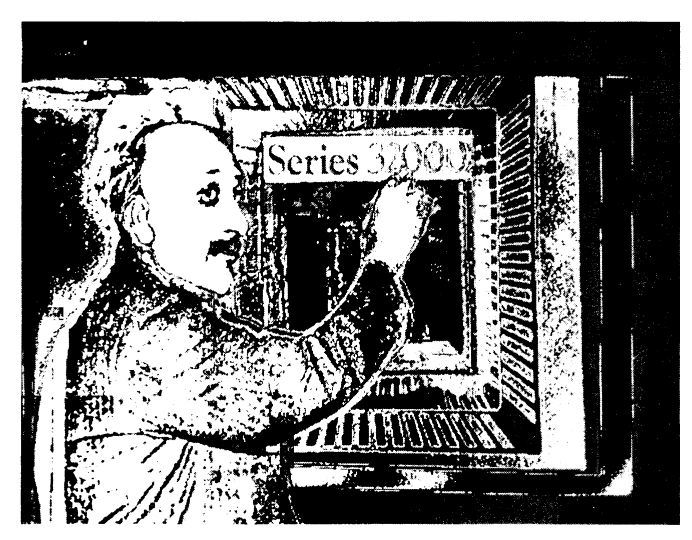
Inhalt

1986
Franzis-Verlag GmbH, Karlstraße 37–41, 8000 München 2.
Bearbeitet von der Redaktion der Zeitschrift ELEKTRONIK. Für den Text verantwortlich: Dipl.-Ing. Peter von Bechen.

Druck: Franzis-Druck GmbH, München. Printed in Germany. Imprimé en Allemagne. ZV-Artikel-Nr. 21802 · F/ZV/986/1222/5' · ISSN 0170-0898

Mick Curry

Mikroprozessoren und Management



(Bild: Kempkens)

Mit dem Aufkommen von 32-Bit-Mikroprozessoren (MPU) ist es notwendig, sich auf die grundlegenden Ideen beim Konzipieren von Produkten auf der Grundlage von solchen Bausteinen zurückzubesinnen, egal ob es sich um 8-, 16- oder 32-Bit-Systeme handelt. Die neuen Prozessoren versetzen einen in die Lage, die Philosophie in bezug auf neue Produktkonzepte zu verändern und einen Einfluß auf die Veränderungen des Marktes der nächsten zehn Jahre zu bewirken. Um zu verstehen, warum solche Veränderungen wünschenswert sind, ist es notwendig, zunächst einmal in die Vergangenheit zu schauen.

Die ersten Anfänge

In den frühen 70er Jahren führte die Einführung von 4- und 8-Bit-MPUs zu einer merklichen Veränderung in den Entwurfsmethoden. Diese neuen Bauelemente boten große Vorteile im Vergleich mit ihren diskret aufgebauten Vorgängern. Ein Entwickler konnte jetzt die Leistung erhöhen, während die Anzahl der Bauelemente und damit die Kosten für die zu entwickelnden Geräte geringer wurden. Der größte Vorteil allerdings ergab sich in bezug auf die Flexibilität. Veränderungen am Gerätekonzept ließen sich mit den neuen Bauelementen ausführen, ohne daß ein Lötkolben erforderlich war – von diesem Zeitpunkt an war Software ein fester Begriff im Vokabular eines Entwicklers.

Vor dieser Zeit waren die Halbleiterhersteller die unangefochtenen Experten in diesem Bereich. OEM-Kunden, die solche Elemente zum Aufbau von Produkten benutzten, und die Endkunden, die diese Geräte und Systeme kauften, spielten nur eine unwichtige Rolle bei der Spezifikation der internen Struktur einer MPU. Dies war den Ingenieuren vorbehalten. Die Produkte, die im Bereich der Endkunden verfügbar wurden, kamen aus den Entwicklungsabteilungen und nicht aus Marketingabteilungen. Ingenieure, die ein kleines Labor in der heimischen Garage hatten, legten die Grundsteine für große Firmen. Management und Marketing orientierten sich an den „Zauberkünsten" der Ingenieure und versuchten, auf dieser Welle zum Erfolg zu schwimmen.

Die Zeiten änderten sich

In den späten 70er Jahren änderte sich die Szene. Die „schwarze Kunst" der Mikrocomputerentwicklung war nicht nur bei OEM-Anbietern bekannt, sondern auch für Endkunden kein Buch mit sieben Siegeln. Es gab genug Leute in beiden Bereichen, die jetzt das Aussehen der internen Strukturen zukünftiger Prozessoren mitbestimmen wollten. Zu dieser Zeit waren 16-Bit-MPUs Stand der Technik. Dabei handelte es sich um große Brüder der ersten 8-Bit-Generation. Einige Hersteller sagten seinerzeit voraus, daß Personal Computer sich in allen Bereichen der Geschäftswelt durchsetzen würden, und sogar in den heimischen Bereich über kurz oder lang vordringen.

Mit diesem neuen Verständnis der Technik begannen die Endkunden, mehr Leistung für den gleichen Preis oder sogar noch weniger Geld zu verlangen. Dies brachte für OEM-Anbieter große Probleme:

- Eine zunehmende Leistung der Produkte bedeutete auch eine kürzere Lebensdauer (in der Tat waren viele Produkte ein Mißerfolg, weil sie in bezug auf die Leistung bereits überholt waren, bevor sie auf den Markt kamen).
- Zunehmende Leistung bedeutet auch, daß die Entwicklung wesentlich komplexer ist und längere Zeit in Anspruch nimmt.

Das Problem einer kürzeren Lebensdauer eines Produktes, das eine längere Entwicklungszeit erfordert, aber

das gleichzeitig vom Kunden nicht durch einen höheren Preis honoriert wird, bedeutete für viele Unternehmen den sicheren Untergang. Dieser Widerspruch konnte mit existierenden MPU-Familien nicht gelöst werden. Diese litten nämlich unter zwei wichtigen Problemen in ihrer Struktur:

- Um Software schnell entwickeln zu können, ist es notwendig, eine höhere Programmiersprache (HLL) zu benutzen. Diese Programme sind in einer Form geschrieben, die sehr eng mit normaler (englischer) Sprache verwandt ist. Damit ist sie leicht verständlich. Die interne Struktur von traditionellen 8- und 16-Bit-MPUs ließ eine effiziente Benutzung einer HLL nicht zu. Um mehr Leistung erreichen zu können, ist es notwendig, auf Maschinencode-Ebene zu arbeiten (eine numerische Sprache, die auf Bitebene abläuft), allerdings ist das außerordentlich kompliziert und für die Entwicklung von Software viel zu langwierig. Als Kompromißsprache zieht man daher den sogenannten „Assembler" heran. Obwohl es wesentlich einfacher ist, in einem Assembler zu arbeiten als mit einer Maschinensprache, ist es trotzdem immer noch sehr kompliziert und erfordert lange Entwicklungszeiten sowie großen Aufwand bei der Fehlersuche. Größere Verzögerungen können entstehen, wenn ein Programmierer das Unternehmen verläßt oder ein altes Programm neu geschrieben werden muß.
- Um die Kompatibilität zwischen 8- und 16-Bit-Elementen einhalten zu können, wurden gewisse Vereinfachungen in der Struktur erforderlich. Dies wiederum ermöglichte die Portierbarkeit eines Programmes von einem 8- auf ein 16-Bit-Bauelement. Allerdings läßt sich in den meisten Fällen dieser Weg nicht umgekehrt beschreiten, d. h. 16-Bit-Programme lassen sich in der Regel nicht auf 8-Bit-Prozessoren übertragen. Ein weiterer Vorteil dieser Vereinfachungen war, daß existierende 8-Bit-Entwicklungssysteme für die 16-Bit-Prozessoren erweitert werden konnten. Allerdings war der Preis für die Kompatibilität die Tatsache, daß die älteren Prozessorarchitekturen die neueren MPU-Konzepte stark beinflußten.

Die neue Technologie steht zur Verfügung

Am Ende der 70er Jahre gab es wichtige Fortschritte im Bereich der Siliziumstrukturen. CMOS begann sich mit einer geringen Leistungsaufnahme zur dominierenden Technologie zu entwickeln. Die Geometrie und die Chipstrukturen konnten verkleinert werden, während die Abmessungen des Siliziumwafers zunahmen. Diese Faktoren zusammen erschlossen den Chipherstellern neue Dimensionen:

- Es konnten komplexere Chips produziert werden, die geringere Leistung aufnehmen,
- diese paßten in kleinere Gehäuse,
- die erhöhte Produktausbeute machte sich im reduzierten Preis bemerkbar.

Zu dieser Zeit sagte man voraus, daß in den frühen 80er Jahren die Chiphersteller ihren Kunden eine völlig neue Art Mikroprozessoren anbieten könnten. Die Frage war, ob man die derzeit zur Verfügung stehende MPU-Architektur noch einmal verbessern oder statt dessen eine völlig neue Struktur entwickeln sollte.

In der Zwischenzeit machte auch die Computertechnik Fortschritte. Universelle Betriebssysteme wie z. B. UNIX, die Programmiersprache Pascal sowie die Voraussetzungen für künstliche Intelligenz kristallisierten sich als mögliche zukünftige Standards heraus. Es entstand ein Bedarf an billigen, großen, uniformen Speichern, Computern mit symmetrischer Architektur und Fließkomma-Recheneinrichtungen. Bis zu diesem Zeitpunkt waren Minicomputer und Großrechner das angestammte Gebiet der Computerwissenschaftler. Mit dem Anwachsen der Komplexität einzelner MPU-Chips wurde es allerdings ganz offensichtlich, daß nun die Zeit gekommen war, daß auch Computer dieser Größenordnung die Ansprüche aus diesem Bereich erfüllen können.

Der Kunde ist immer König

Zur Jahreswende 1978/79 sprachen Marktforscher von National Semiconductor mit Benutzern und OEM-Kunden, welche Merkmale sie für zukünftige MPUs wünschten, die ihre Anforderungen bis zum Jahr 2000 erfüllen könnten. Dabei ergab sich folgende übereinstimmende Meinung:

- Verbesserte Software-Produktivität. Hiermit ergibt sich eine verkürzte Zeit bis zum Erreichen der Marktreife, wobei die Kosten für Software- und Hardware-Entwicklung fallen.
- Architektur mit Langzeit-Kompatibilität. Diese Eigenschaft schützt alle Investitionen für die Software.
- Effiziente Verwaltung von großen Speicherbereichen. Die Programme werden immer komplexer und benötigen immer mehr Speicherkapazität. Der Zugriff auf diese größeren Speicherstrukturen darf den Durchsatz der MPU nicht behindern.

Interessant an diesen Antworten ist, daß alle drei Wünsche Anforderungen der Software entsprechen. Die Verarbeitung von Software hat demnach höchste Priorität. Die existierenden MPU-Familien wurden ursprünglich nicht unter diesem Gesichtspunkt entwickelt. Damit war klar, daß jetzt die Zeit für eine MPU mit einer neuen Architektur reif war. Diese sollte speziell so ausgelegt sein, daß sie die Softwareproduktion erleichtert, wobei alle Möglichkeiten für eine langfristige Leistungserhöhung offengehalten werden, was natürlich unter vollständiger Software-Kompatibilität erfolgen sollte.

Einstieg in die 32-Bit-Welt

Eine Busbreite von 32 Bit ergab sich aus zwei Gründen:

– Architektur:
  Unterstützt höhere Programmiersprachen,
  großer Speicheradreßbereich (4 GByte),
  die Möglichkeit, daß die Fließkommaeinheit (FPU) zehn Dezimalstellen verarbeiten kann,
  Möglichkeiten für zukünftige Erweiterungen (über einen Zeitraum von zehn Jahren).
– Busbreite:
  Ermöglichen von einem erhöhten Datentransfer,
  weniger Zugriffe auf den Speicher,
  Erhöhung des Prozessordurchsatzes.

Verschiedene große OEM-Häuser produzierten ihre eigene MPU, um ihre Anforderungen befriedigen zu können. Die erste kommerziell erhältliche 32-Bit-MPU-Familie wurde 1982 von National Semiconductor vorgestellt. Es handelte sich beim NS32032 um eine echte 32-Bit-MPU mit 32-Bit-Registern, einer 32-Bit-ALU (Arithmetik-Logik-Einheit) sowie einem internen 32-Bit-Datenbus. Das neue Familienmitglied NS32332 verfügt darüber hinaus über 32 externe Adressierleitungen und eine dynamische Busbreiten-Anpassung.

Die Situation heute

Heute, im Jahre 1986, sind 32-Bit-MPUs Stand der Technik. Was bieten sie, das den Entwickler von 8- oder 16-Bit-Systemen dazu verleiten könnte, sie zu verwenden? Die Antwort liegt in den bisher gemachten Ausführungen: Verbesserte Softwareproduktivität und langfristig kompatible Architektur haben nichts mit der Leistung einer MPU zu tun. Die effiziente Verwaltung großer Speicherbereiche dagegen ist direkt von der MPU-Leistung abhängig.

Die Serie 32000 von National Semiconductor bietet diese drei Vorteile auch für den 8- und 16-Bit-CPU-Benutzer, indem drei verschiedene Versionen dieses Bauelementes zur Verfügung stehen. Die interne Struktur dieser drei Prozessoren arbeitet mit einer Breite von 32 Bit; allerdings sind die Datenbusse 8, 16 oder 32 Bit breit. Alle Typen arbeiten mit gleicher Software, allerdings mit unterschiedlicher Leistung.

Eine 16-Bit-MPU hat eine geringere Leistung, weil im Vergleich zur 32-Bit-Version mehr Speicherzugriffe zum Transfer von Daten erforderlich sind. Die 8-Bit-Ausführung bietet noch weniger Durchsatzkapazität als das 16-Bit-Bauelement. Weil die Durchsatzmöglichkeit eine direkte Funktion der Datenbusbreite ist, findet man bei allen anderen Familien von 8- und 16-Bit-Familien die gleichen Begrenzungen. Allerdings sorgt die interne 32-Bit-Architektur der neuen MPUs für eine erhöhte Verarbeitungsleistung. Dadurch ist es möglich, daß sie Vorteile gegenüber Bauelementen bieten, die schon längere Zeit auf dem Markt angeboten werden und außerdem die verbesserte Softwareproduktivität und Langzeitkompatibilität bieten. Vier weitere Chips runden die Prozessorfamilie ab: die Fließkommaeinheit NS32081, die Speicherverwaltungseinheit NS32082, die Zähler-/Zeitgeber-Einheit NS32201 sowie die Interrupt-Controllereinheit NS32202.

Was bringt's?

Was kosten uns alle diese „tollen" Eigenschaften? Die Chipgröße der Prozessoren NS32008 und NS32016 ist, aufgrund der 32-Bit-Architektur, größer als diejenige typischer 8- oder 16-Bit-MPUs. Konsequenterweise sind sie auch teurer als die üblichen Typen dieser Klassen. Die 8-Bit-Ausführung kostet etwa um den Faktor 1,5...2 mehr als vergleichbare Typen, die 16-Bit-Ausführung liegt etwa um den Faktor 1...1,5 über anderen Versionen.

Diese zusätzlichen Kosten für das Silizium werden durch Einsparungen bei der Entwicklung mehr als kompensiert, außerdem durch die Flexibilität, die von solchen Familien geboten wird. Um die Flexibilität zu zeigen, kann man sich folgendes Beispiel vorstellen: Ein Produkt soll auf der Grundlage einer 16-Bit-MPU entwickelt werden. Diese soll allgemeine Computerfunktionen bieten. Außerdem sollen leistungsfähige Arithmetikeigenschaften für ingenieurmäßige Anwendungen zur Verfügung stehen oder Zugriffe auf große Dateien für die Sortierung von Daten möglich sein. Es bestehen folgende Möglichkeiten:

1. Eine 16-Bit-MPU, bei der die gesamten Speicherzugriffe und alle FPU-Funktionen mit Hilfe von Software erfolgen.

2. Eine 16-Bit-MPU und eine FPU, bei denen alle Arithmetikfunktionen von der FPU-Hardware ausgeführt werden.

3. Eine 16-Bit-MPU und eine MMU, bei denen alle Speicherverwaltungsfunktionen von der MMU-Hardware übernommen werden.

4. Eine 16-Bit-MPU sowie eine FPU und eine MMU, wobei alle Arithmetik- und Speicherverwaltungs-Aufgaben von der Hardware erledigt werden.

Die Software für diese hier aufgezählten Möglichkeiten ist identisch. Der Prozessor prüft, welche Hardware vorhanden ist, bevor er eine Speicher- oder Arithmetik-Routine ausführt. Falls vorhanden, übernehmen die Slave-MMU oder -FPU die Kontrolle des Systems und führen die entsprechenden Funktionen für die MPU aus. Im anderen Fall muß die MPU diese Funktionen übernehmen. Insgesamt ergeben sich die hier gezeigten vier Optionen mit unterschiedlicher Leistung, aber auf Grundlage einer einzigen MPU.

Soll das Produkt in einer niedrigeren Leistungsklasse angesiedelt sein, läßt sich auch problemlos die 8-Bit-Version der MPU verwenden. Im umgekehrten Fall, wenn das Endprodukt im oberen Leistungsbereich angesiedelt sein soll, gibt es keine Probleme, mit Hilfe einer

32-Bit-MPU eine Leistungssteigerung vorzunehmen. Auch hier bleibt die vorhandene Software brauchbar.

Sowohl die FPU als auch die MMU sind Slave-Prozessoren. Dieses Konzept bedeutet, daß sie Erweiterungen der MPU-Architektur darstellen. Sie erfordern keine Handshake-Steuerung wie ein separater Coprozessor, so daß keine Prozessorzeit zur Steuerung verschwendet wird. Die Architektur läßt sich vom Benutzer problemlos erweitern, wenn er entscheidet, seine eigenen Slave-Prozessoren zu installieren. Diese kundenspezifische Auslegung eines Systems bietet auch die Möglichkeit, spezielle Instruktionen zu realisieren. Der kundenspezifische Slave-Prozessor kann beispielsweise aus diskreten Logikbausteinen bestehen. Möglich ist auch die Verwendung von Gate-Arrays oder Standard-Zellen-Bausteinen, so daß es für den Mitbewerb schwierig ist, Systeme zu kopieren.

Die hier beschriebene Struktur bedeutet auch, daß National Semiconductor Erweiterungen an der Architektur der Prozessorfamilie vornehmen kann, damit die langfristige Daseinsberechtigung dieser Familie gesichert ist.

Kosten und Investitionen

Es gibt sicher Anwendungen, bei denen die Produktionsvolumina so groß sind, daß die Entwicklungskosten pro fertiggestellter Einheit sehr gering sind. Konsumprodukte sind typisch für diese Kategorie. Hier achtet man auf den Preis der einzelnen Bauelemente. Die Anzahl der bei folgenden Generationen erforderlichen Teile ist beschränkt. Damit wird auch der langfristige Wert von Software und Entwicklungsaufwand reduziert.

Wer braucht 32-Bit-Prozessoren?

Als 1976 die ersten 16-Bit-Prozessoren auf den Markt kamen, waren sich die meisten Fachleute einig, daß es nur fünf Jahre dauern würde, bis der 16-Bit-Markt den 8-Bit-Markt überflügelt. In Wirklichkeit geschah das jedoch weder nach fünf Jahren noch nach neun. 1984 hatten 16-Bit-Mikroprozessoren an den gesamten µP-Lieferungen nur einen Anteil von 10 %, und heute sagen Kenner der Verhältnisse, daß es noch weitere fünf Jahre dauern kann, bis der 16-Bit-Markt das Volumen des 8-Bit-Marktes erreichen wird. Einige der Beobachter, die rückblickend die Marktentwicklung der Prozessoren mit 16 Bit viel zu optimistisch vorhersagten, reagierten auf die tatsächliche Entwicklung mit recht pessimistischen Prognosen für die 32-Bit-Bausteine. Es grenzt fast an Ironie, daß zu einer Zeit, in der sich die Halbleiterfirmen zum Kampf um die Anteile im 32-Bit-Markt rüsten, die meisten ausgelieferten Mikroprozessoren Typen mit 8 Bit sind. Die Anwender sind bis heute nicht voll auf 16 Bit übergeschwenkt, wer also braucht 32 Bit?

Das Datenformat ist nur ein Aspekt

Es gibt eine Überlegung, nach der heutzutage schwerlich jemand wirklich einen Prozessor mit 32 Bit braucht. Dieses simple Argument beruht teils auf historischen Erfahrungen, teils auf der Art von Daten, die in Mikroprozessor-Systemen am häufigsten vorkommen.

Hier seien zuerst die verschiedenen Arten von Daten betrachtet, die Mikroprozessoren verarbeiten, allen voran die numerischen. Eine CPU mit 8 Bit kann direkt im 2er-Komplement codierte ganze Zahlen im Bereich −128...+127 oder positive Zahlen 0...255 verarbeiten. Das ist im allgemeinen nicht ausreichend, so daß die meisten 8-Bit-Anwender Arithmetik mit doppelter Präzision nutzen, um ganze Zahlen mit 16 Bit zu bearbeiten. Das bedeutet, daß die Programmcodes länger und die Ausführungszeiten langsamer werden. Doch für viele Anwendungen, bei denen 8-Bit-CPUs verwendet werden, ist die erzielbare Leistungsfähigkeit befriedigend.

Eine 16-Bit-CPU kann 65 536 ganze Zahlen verarbeiten, ein Umfang, der vielen Anwendern durchaus ausreicht. Wenn eine größere numerische Genauigkeit gefordert wird, ist es oft besser auf Gleitkommazahlen überzugehen, vor allem seit es schnelle Gleitkomma-Prozessoren wie den Typ NS32081 gibt, die leicht mit 16-Bit- und 32-Bit-CPUs zusammenschaltbar sind. Entsprechend der vom IEEE vorgeschlagenen Norm werden Gleitkommazahlen in einem 32-Bit- oder einem 64-Bit-Format dargestellt, doch ob diese Bitbreiten mit der Wortlänge der CPU übereinstimmen, ist nicht relevant.

Viele Mikroprozessoren wenden einen großen Teil ihrer Zeit für die Verarbeitung von Daten auf, die eigentlich Texte in Form von ASCII-Zeichen darstellen und Byte für Byte bearbeitet werden. Auch die E/A-Anforderungen können normalerweise mit 8-Bit-Ports erfüllt werden, und selbst die kritischsten Bedürfnisse der Datenerfassung befriedigen Datenumsetzer mit 16 Bit.

Historische Betrachtungen unterstützen das oben genannte Argument. Die ersten µP-Familien hatten 4-Bit-CPUs und wurden benutzt, weil nichts anderes verfügbar war. Als 8-Bit-CPUs auf den Markt kamen, lösten sie die 4-Bit-Typen schnell ab, da 8 Bit ein wesentlich angenehmeres Datenformat ist und die größere Computerleistung sinnvoll genutzt werden konnte.

Das Erscheinen von 16-Bit-Mikroprozessoren hat nicht das gleiche enthusiastische Echo hervorgerufen. Für viele Anwender, vor allem für jene, die mit Textverarbeitung befaßt sind, sind Bytes auch weiterhin das wichtigste Datenformat, und die höhere Leistungsfähigkeit von 16-Bit-Typen ist oft kein entscheidender Faktor. Die Erweiterung dieses Arguments ist naheliegend: Wenn 16 Bit bereits ein „Overkill" sind, müssen 32 Bit es erst recht sein.

Wie viele simple Argumente, enthält auch dieses einen zutreffenden Teil, der einen viel schwerer wiegenden Fehlschluß überdeckt. Es ist richtig, daß die meisten CPUs, sogar die 32-Bit-Typen, viel Zeit für die Verarbeitung von Daten

Allerdings ist zu beachten, daß die meisten Produkte eine permanente Evolution durchmachen. Sicherheitssysteme, Telecomprodukte, Computer, industrielle Steuerungen usw. werden in ihrer Leistung gesteigert. Was man heute für die Software ausgibt, kann unter diesen Umständen eine Investition für die Zukunft sein.

Eine Management-Entscheidung

MPUs haben einen Entwicklungsstand erreicht, bei dem es schwierig ist, eine Auswahl nur aufgrund der Leistung zu treffen. Die Entscheidung, welche MPU-Familie Verwendung finden soll, läßt sich mittlerweile nur auf der Grundlage des strategischen Marketing und wirtschaftlicher Gesichtspunkte fällen. Neben den Ingenieuren sollten das obere Management, das Marketing, die Software-Entwickler und die Verkäufer diese Entscheidung mittragen. Die Auswahl betrifft nämlich wichtige Einflüsse des Geschäftes:

- die langfristigen Anforderungen des Marktes,
- die derzeitigen Entwicklungen und zukünftigen Trends, um die langfristigen Anforderungen erfüllen zu können,
- die Lebensdauer eines Produkts,
- das erwartete Absatzvolumen pro Jahr (heute und in Zukunft),
- Entwicklungszeit und -kosten (heute und in Zukunft),
- Amortisierung von Software-Entwicklungskosten,
- die Position im Mitbewerb und die Angriffsmöglichkeiten der Konkurrenz in kurzen und längeren Zeiträumen.

aufwenden, die auch mit 8 Bit umfassenden Bytes oder 16-Bit-Worten dargestellt werden können. Doch das Format der Daten ist nur ein Aspekt beim Betrieb von Mikroprozessoren und oft ein relativ unbedeutender.

Zugriff und Programmcode sind wichtig

Der Wert einer 32-Bit-Architektur hat mit der Art von Daten, die zu verarbeiten sind, nichts zu tun. Die wichtigen Fragen sind heute, wie auf Daten zugegriffen wird und wie der Programmcode geschrieben ist. Läßt man die Arten der Daten außer acht und betrachtet statt dessen, wie zu ihnen zugegriffen wird, ergibt sich ein ganz anderes Bild. Ausführliche Untersuchungen haben gezeigt, daß eine CPU genau so viel Zeit aufwendet, Daten aus dem Speicher zu holen und wieder abzuspeichern, wie sie für die Verarbeitung der Daten braucht. Bei Systemen mit relativ wenig RAM und nur einfachen Adressierungsarten beim Programmcode sind typische 8-Bit-CPUs ausreichend.

Wenn jedoch die RAM-Kapazität eines Systems 64 KByte übersteigt oder wenn Datenstrukturen wie Arrays oder Records benötigt werden, braucht man eine effiziente Architektur. Betrachtet sei als Beispiel der Zugriff zu einem Element eines mehrdimensionalen Arrays, das in einem RAM mit 1 MByte abgelegt ist: Als erstes ist, wenn man nicht zu der veralteten und höchst ineffizienten Technik der Bank-Umschaltung greift, ein Adressenzeiger von 20 Bit notwendig, um 1 MByte zu adressieren. Um die Adresse des gesuchten Elements zu berechnen, müssen mehrere Multiplikationen und Additionen zur Berücksichtigung der Array-Dimensionen ausgeführt werden.

Es ist klar erkennbar, daß 32-Bit-Arithmetik erforderlich ist, um die Adressen effizient zu bearbeiten, gleichgültig ob die gesuchten Daten Bytes, Worte oder von anderer Länge sind. In der Tat ist eine gute 32-Bit-Architektur symmetrisch und orthogonal, was jede Operation mit jedem relevanten Datentyp und jeder beliebigen Adressierungsart zuläßt.

Höhere Programmiersprachen werden unterstützt

Die wichtigste Eigenschaft heutiger fortschrittlicher CPUs ist nicht der Umstand, ob sie eine 32-Bit-ALU haben, sondern ob sie für die Unterstützung höherer Programmiersprachen optimiert sind. Die Erstellung der Software in einer höheren Sprache spart Zeit und Kosten; die Reduzierung der Entwicklungskosten und die schnellere Marktreife können von entscheidender Bedeutung sein. Der einzige Gesichtspunkt, der gegen eine weitere Verbreitung höherer Programmiersprachen steht, ist die Erfahrung, daß Compiler bis zum Dreifachen längere Objekt-Codes erzeugen, als erfahrene Programmierer in Assembler-Sprache.

In Anbetracht der immer preiswerter werdenden Hardware und der zunehmenden Knappheit an guten Programmierern wird die Effizienz von Compilern ein wesentlicher Faktor bei der Auswahl eines Mikroprozessors. Es gibt wichtige Gründe, warum Architekturen für effiziente Programmiersprachen-Unterstützung auf 32-Bit-ALUs basieren, und alle heute erhältlichen 32-Bit-CPUs wurden – mit unterschiedlichem Erfolg – für die Unterstützung höherer Programmiersprachen ausgelegt.

Das sind die Gründe, warum historische Vergleiche irreführend sind. Die 16-Bit-Mikroprozessoren waren nicht wesentlich leistungsfähiger als die 8-Bit-Typen und hatten kein wesentlich anderes Konzept. Zwischen CPUs mit 16 Bit und mit 32 Bit gibt es jedoch fundamentale Unterschiede. Obwohl man im allgemeinen nach der Bitbreite der ALU klassifiziert, liegt der wirkliche Unterschied in der Architektur.

Zusammenfassend läßt sich sagen, daß 32-Bit-Prozessoren eine wesentlich größere Leistungsfähigkeit bieten als 8-Bit- oder 16-Bit-Typen, viele Anwender dieses Leistungsniveau aber nicht brauchen. Sie ermöglichen jedoch, neue Produkte schneller und kostengünstiger auf den Markt zu bringen; und wer kann eigentlich behaupten, daß er das nicht braucht?

Dr. Werner Trattnig

Dies sind nur einige der möglichen Fragen, die sich an der Art des Produktes unterscheiden können, aber hier als Beispiel dafür dienen, welche Konsequenzen die Entscheidung auch für Nicht-Ingenieur-Bereiche eines Unternehmens hat.

Die Zukunft

Die Serie 32000 wird in der Leistung immer mehr gesteigert werden (der Typ NS32332 aus dem Jahr 1985 ist bereits ein Bauelement der 2. Generation, das Super-Mini-Architektur und -Leistung bietet). Hierbei wurde insbesondere auf die Kompatibilität geachtet. Moderne Gehäusetechnik, z. B. das Pin-Grid-Array (PGR) sowie Typen für Oberflächenmontage werden den Standard neben dem DIL-Gehäuse bilden.

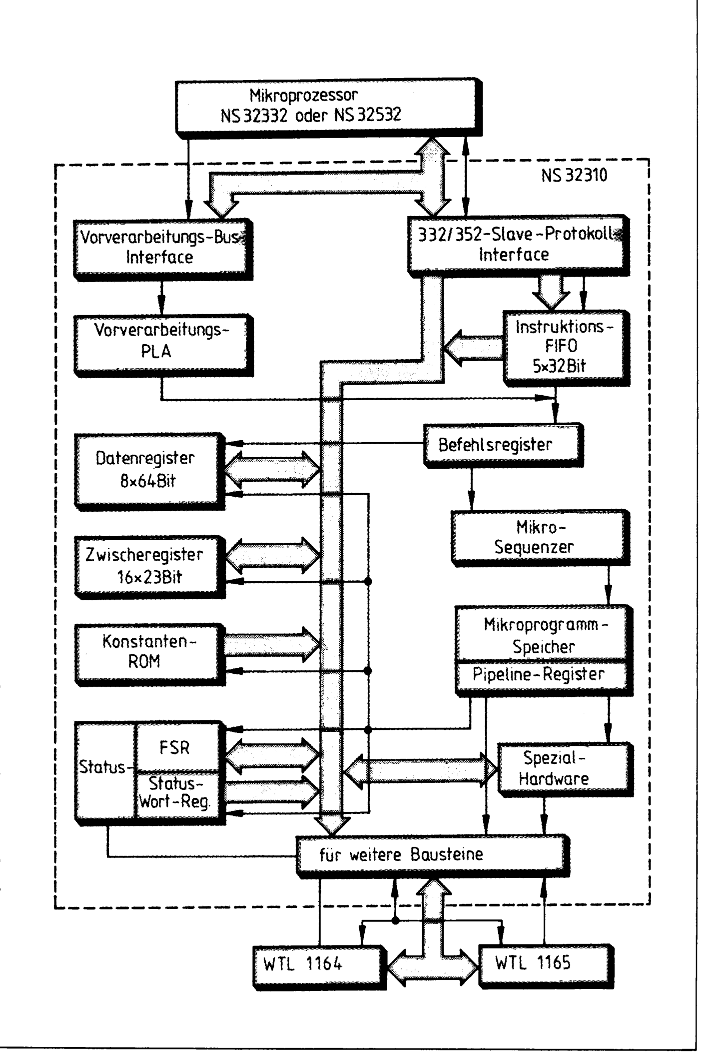
Was ist bei geringeren Busbreiten zu erwarten?

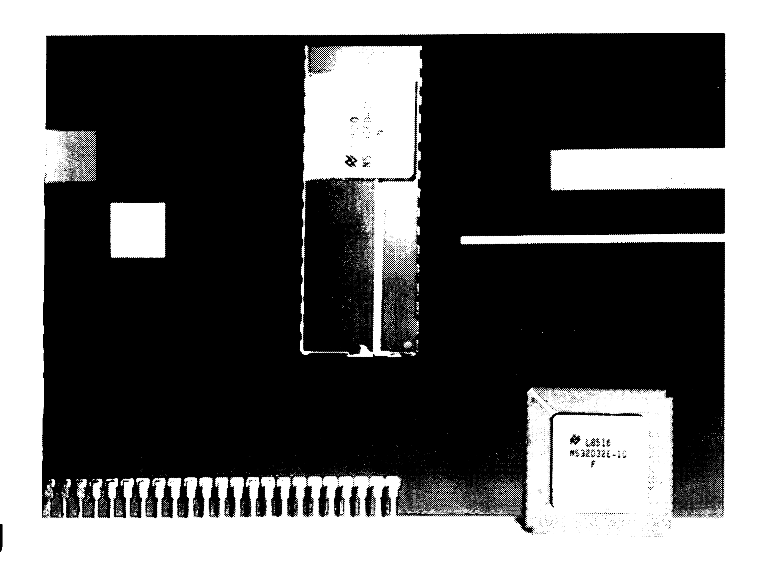
Für die überschaubare Zukunft gibt es einen Bedarf für alle MPU-Größen, ob es nun Typen mit 4, 8, 16 oder 32 Bit sind. In der Praxis wird das Volumen der verkauften Bauelemente umgekehrt proportional zur Datenbusbreite sein, das heißt, es werden mehr 4- als 8-Bit-Bauelemente verkauft. Dies liegt daran, daß der kleinere Prozessor den größeren Markt hat. Die gleichen Beziehungen bestehen zwischen 8- und 16- bzw. 16- und 32-Bit-Systemen.

Die kleineren Bauelemente (4 und 8 Bit) sowie die dazugehörige Logik wird eines Tages fast ausschließlich in Form von Gate-Arrays zur Verfügung stehen, wenn die entsprechenden Bibliotheken auch komplexe Funktionen enthalten. Die größeren Bauelemente (16 und 32 Bit) werden vielleicht diesem Trend auch folgen, wenn Gate-Arrays groß genug sind oder, viel wahrscheinlicher, die Standard-Zellen-Technik ihren Durchbruch erreicht. Falls dies geschieht, wird das keinen Einfluß auf die Software haben. Was immer auch geschieht, für den größten Teil der Mikroprozessoranwender bietet sich mit der 32000-Familie eine Möglichkeit, in ihre langfristige Zukunft zu investieren und dabei die Anforderungen heutiger Systeme erfüllen zu können.

Fließkomma-Arithmetikeinheit NS 32310

Der Hochleistungsprozessor für Gleitkommaberechnung NS 32310 gehört zur 32000-Familie. Dieser Baustein ist vollständig Softwarekompatibel zum Gleitkommaprozessor NS 32081. Ermöglicht jedoch bei gleicher Taktfrequenz einen etwa drei-bis fünf Datendurchsatz bei den Grundrechenarten. Diese bemerkenswerte Leistungssteigerung wurde möglich, da die eigentliche Rechenarbeit in den nachgeschalteten Super-Gleikommaprozessoren WTL 1164 und WTL 1165 von Weitek ausgeführt werden.
Diese Konfigurationen erlaubt es, vorhandene Software, die auf einem System mit der CPU NS 32016 plus FPU NS 32081 geschrieben wurde, auf ein System mit der CPU NS 32332 und FPA NS 32310 zu übertragen und ohne Änderung ablaufen zu lassen.
Neben den von der NS 32081 her bekannten Funktionen bietet der NS 32310 zusätzlich den vollen Befehlsumpfang IEEE 754 Fassung 10.0. Dies bedeutet, daß auch die trigonometrischen Funktionen wie sin, cos, tan, usw. in Hardware und damit wesentlich schneller ausgeführt werden. Mit diesem FPA (Floating Point Accelerator) stößt die 32000-Mikroprozessor-Familie in den Bereich von 3,5 MFlops vor, was bisher Systemen vorbehalten war, die 100 000 DM und mehr gekostet haben. Bei einem Preis von etwa DM 1200.- für die drei Chips (NS 32310, WTL 1164 und WTL 1165) ist die Anwendung in Grafikprozessoren und Industrie-Roboter preiswerter möglich.
Den Prozessor wird von National Semiconductor im Hochleistungs-CMOS-Prozeß M²CMOS hergestellt und benötigt nur eine einfache 5-V-Spannungsversorgung. Zusammen mit der CPU NS 32332 arbeitet der Chipsatz bis zu einer Taktfrequenz von 15 MHz.
Uwe Krause





Uwe Krause

Die 32000-Familie

Architektur und Implementierung

Seit 1971, als der erste Mikroprozessor entwickelt wurde, sind drei unterschiedliche Generationen von Maschinen auf dem Markt erschienen. Zuerst kamen die 4-Bit-Mikroprozessoren (z. B. der Typ 4004 von Intel), danach die 8-Bit-Mikroprozessoren (z. B. 8080, Z80 und 6800), anschließend die 16-Bit-Maschinen (z. B. 8086, Z8000 und 68000). Diese Prozessorgenerationen unterscheiden sich in verschiedenen Punkten, die sich alle auf die Anzahl der Datenbits beziehen, die eine solche Maschine im Parallelbetrieb verarbeiten kann. Entscheidend ist aber der architektonische Aufbau, der die Leistung bestimmt.

Beispielsweise sind alle 8-Bit-Mikroprozessoren in der Lage, zumindest 8-Bit-Zahlen zu addieren oder zu multiplizieren. Deshalb weist der Bus (die Verbindungsleitungen, über die Daten von oder zum Prozessor gebracht werden) eine Breite von 8 Bit auf. Aus ähnlichen Gründen haben die Register in diesen Maschinen ebenfalls eine Breite von 8 Bit.

Trotz einiger Ausnahmen kann man davon ausgehen, daß alle Strukturen in 8-Bit-Maschinen auch eine Breite von 8 Bit haben. Ähnlich ist es bei 16-Bit-Maschinen. Beispiel für eine Ausnahme ist, daß die Adressen bei 8-Bit-Maschinen in den meisten Fällen in einer Breite von 16 Bit vorliegen, weil 8 Bit lediglich 256 Byte Speicher adressieren könnten, was viel zu wenig für den Programmumfang solcher Maschinen ist. Aus diesem Grund ist es manchmal nicht ganz einfach zu unterscheiden, ob eine bestimmte Maschine ein 8-Bit- oder ein 16-Bit-Mikroprozessor oder gar ein Typ mit einer ganz anderen Wortbreite ist. So wird beispielsweise der Prozessor 8088 von Intel manchmal als 8-Bit-Maschine bezeichnet, weil der externe Datenbus eine Breite von 8 Bit aufweist. Richtiger ist allerdings die Bezeichnung 16-Bit-Mikroprozessor, weil dieselben Programme wie beim 16-Bit-Typ 8086 ausgeführt werden.

In vergangenen Jahren, insbesondere seit der Einführung der 32000-Familie, wurde die Verwirrung bei der Typenbezeichnung noch größer, weil es viele Diskussionen darüber gab, welche Eigenschaften einen Prozessor zu einer 32-Bit-Maschine machen. Diskussionen gab es um die Verarbeitungsleistung und Eigenschaften einzelner Mikroprozessortypen. 32-Bit-Mikroprozessoren sind leistungsfähiger als 16-Bit-Prozessoren, ähnlich wie 16-Bit-Prozessoren leistungsfähiger als 8-Bit-Maschinen sind. Allerdings ist die Leistung eines Computers eine sehr komplizierte Angelegenheit, und die Zurückführung auf einzelne Schlagwörter wie „32 Bit“ oder „16 Bit“ kann nicht alle Merkmale, die wichtig sind, berücksichtigen.

In diesem Beitrag soll anhand einiger derzeit auf dem Markt erhältlicher Mikroprozessoren gezeigt werden, welche Punkte wichtig sind, damit man die für Leistung bestimmenden Faktoren verstehen kann. Dazu müssen zwei grundsätzliche Punkte des Konzeptes von Mikroprozessoren erläutert werden: die Architektur eines Computers und deren Implementierung. Unter Architektur versteht man das Software-Interface einer Maschine auf Objektcode-Ebene. Die Implementierung ist andererseits die tatsächliche Realisierung der Architektur in Hardware. Zwei Maschinen können beispielsweise gleiche Architektur aufweisen, wenn Software auf der einen genausogut wie auf der anderen abläuft, auch wenn die tatsächliche Hardwareausführung der beiden Maschinen völlig unterschiedlich ist.

Eine bestimmte Architektur kann verschiedene Implementationen haben, je nach zur Verfügung stehender Technologie und deren Kosten. Beispielsweise haben alle Mikrocomputer der VAX-Familie eine Architektur, die Bandbreite der Technologie reicht aber vom MOS-Mikroprozessor bis zur bipolaren Hochgeschwindigkeits-Logik. Software, die für ein beliebiges Mitglied der VAX-Familie geschrieben ist, läuft auf jedem der Mitglieder. Auch Programme, die für den Typ 11/785 geschrieben sind, dem leistungsfähigen Familienmitglied, laufen auf der microVAX, der kleinsten Version dieses Computers. In ähnlicher Weise haben alle Minicomputer der PDP-11-Familie ein und dieselbe Architektur, allerdings besteht ein Unterschied zur Architektur der VAX-Familie.

Wenn man die VAX- und PDP-11-Familien vergleicht, läßt sich erklären, wie Architektur und Implementierung die Computerleistung beeinflussen. Die VAX ist ein klassisches Beispiel für eine 32-Bit-Architektur,

während die PDP-11 die klassische 16-Bit-Architektur darstellt. Aufgrund der architektonischen Merkmale ist die VAX im Vergleich zur PDP-11 die leistungsfähigere Maschine. Allerdings stimmt es nicht, daß jedes Mitglied der VAX-Familie leistungsfähiger als jedes Mitglied der PDP-11-Familie ist. Die PDP-11/70 stellt sich bei vielen Benchmark-Tests als leistungsfähigere Maschine als die microVAX oder die VAX-11/730 heraus. Die Differenzen liegen ganz offensichtlich in der Implementierung. Daraus kann man erkennen, welche wichtige Rolle die Implementierung bei der Bestimmung der Leistung spielt.

Um die komplexen Zusammenhänge in bezug auf die Leistungsbestimmung eines Mikroprozessors zu beleuchten, soll dieser Beitrag die Einzelheiten der Architektur und Implementierung zeigen, die berücksichtigt werden sollen, wenn man eine 32-Bit-Maschine untersucht. Jeder einzelne Punkt wird anhand der 32000-Familie und anderen Mikroprozessoren dargestellt.

Programmiermodell

Der prinzipielle Nachteil von 16-Bit-Architekturen war bisher deren begrenzter Adreßbereich. Der Adreßbereich eines Computers ist durch die Menge der einzelnen Speicherplätze definiert, die sich adressieren lassen. Dieser Wert wird durch die Breite des Programmzählers (PC-Register) bestimmt, der die Adresse der gerade ausgeführten Instruktion enthält. Bei allen 16-Bit-Computern, beginnend beim legendären Whirlwind von 1947 über die Minicomputer „NOVA" von Data General und PDP-11 von DEC in den 70er Jahren und den 16-Bit-Mikroprozessoren wie z. B. 8002, hatte der PC eine Breite von 16 Bit. Daher können diese 16-Bit-Maschinen lediglich 2^{16} (das heißt 65 536) Speicherplätze adressieren.

Dies veranlaßte Ende der 70er Jahre sowohl die Firmen DEC als auch Data General, 32-Bit-Minicomputer zu entwickeln, so daß man die Beschränkungen des 16-Bit-Adreßbereiches überwinden konnte. Die gleiche Motivation war auch wichtiger Grund für die Entwicklung von 32-Bit-Mikroprozessoren, z. B. für die Serie 32000. Man muß dabei berücksichtigen, daß beim Überschreiten der 16-Bit-Grenze nicht unbedingt die Notwendigkeit besteht, einen Programmzähler mit der vollen Breite von 32 Bit zu benutzen. Immerhin können 32 Bit 2^{32} (das heißt mehr als 4 Mrd.) Speicherplätze adressieren, was wesentlich mehr ist, als die derzeitigen Anwendungen erfordern. Obwohl 32-Bit-Architekturen in der Regel auch einen 32-Bit-Programmzähler besitzen, werden bei den meisten derzeitigen Implementierungen nicht mehr als 24 dieser Bits zur Codierung von Adressen herangezogen. (Sowohl die 32000-Familie als auch der Typ MC68000 benutzen in ihren derzeitig verbreiteten Implementierungen 24-Bit-PCs, obwohl die Architektur bis zu 32 Bit erlaubt.)

Neben dem Programmzähler, der den Adreßbereich des Computers festlegt, verfügen die meisten Architekturen über verschiedene andere Register, die als Zwischenspeicher für Adressen und Daten während der Programmausführung benutzt werden. Das Vorhandensein dieser Daten/Adreß-Register ermöglicht schnellere Ausführung der Befehle, denn sie sind üblicherweise in einer Technologie höherer Arbeitsgeschwindigkeit als der Hauptspeicher ausgeführt. Computerarchitekten haben lange darüber diskutiert, welche optimale Breite und Anzahl diese Register haben sollten.

Die Breite dieser Register in bestimmten Computerarchitekturen ist normalerweise mit der Breite des Programmzählers verknüpft. Zum Beispiel haben sowohl die Minicomputer PDP-11 und ECLIPSE 16-Bit-Programmzähler und auch 16-Bit-Universalregister. Allerdings ist das nicht immer der Fall: Zum Beispiel verfügte der Mikroprozessor 8080 über einen 16-Bit-Programmzähler, aber nur über 8-Bit-Register. Viele Architekturen mit 32-Bit-Register haben einen kleineren PC. Die Anzahl dieser Daten/Adreß-Register in einem bestimmten Computer ist grundsätzlich eine Potenz von 2 – typisch: 4, 8, 16 oder 32 – weil Register durch kurze Bitfelder (üblicherweise 2, 3 oder 4 Bit) in den Befehlen adressiert werden. Die NOVA/ECLIPSE-Familie stellt eine klassische 4-Register-Architektur dar; die PDP-11 hat acht Register, die VAX verfügt über 16.

Die optimale Anzahl der Register wird durch die typische Anzahl der lokalen Variablen in einer Prozedur bestimmt, weil der grundsätzliche Zweck von Registern das Ablegen von Variablen ist, wobei ein schneller Zugriff möglich sein muß. Studien durch William Wulf haben gezeigt, daß fünf Variable, die zur Verfügung stehen, mehr als genug für die meisten Anwendungen sind. Weil andererseits alle Registerinhalte gerettet werden müssen, wenn man eine neue Prozedur aufruft, sollte die Zahl der Register nicht zu hoch sein, sonst besteht die Gefahr, daß eine Aufruf-Operation sehr langwierig wird. Aus diesem Grund scheint die Zahl von 8 Registern (die nächsthöhere Potenz von 2, die über dem Wert 5 liegt) optimal zu sein. Die optimale Breite wird durch den Umfang des typischen Daten-Typs bestimmt, der von der Maschine manipuliert werden soll. Im nächsten Abschnitt wird gezeigt, daß dieser Wert bei einer echten 32-Bit-Maschine auch 32 Bit beträgt.

Neben der Anzahl und der Breite gibt es noch andere wichtige architektonische Merkmale bei Registern, z. B. die Frage, wie universell sie benutzbar sind. Sollte es allen Befehlen erlaubt sein, alle Register frei für jegliche Art von Daten oder Adressen zu benutzen oder sollten bestimmte Register für spezielle Operationen vorbehalten sein? Die optimale Lösung liegt wohl darin, einen bestimmten Umfang der Register für Fließkommazahlen zu verwenden (aufgrund der speziellen Anforderungen), darüber hinaus aber alle Adreß-/Daten-Register für universelle Verwendung zuzulassen. Die Benutzung von universellen Registern anstelle von Spezial-Daten- und Adreß-Registern (wie z. B. beim 68000) ist vorzuziehen, weil dadurch unnötige Register-Register-Transfers über-

flüssig werden, wenn Arithmetikoperationen an Adreßinformationen auszuführen sind.

Die Serie 32000 ist so konzipiert, daß sie über ein Spektrum von Registern mit optimaler Anzahl und Größe verfügt: Es handelt sich um acht 32-Bit-Universalregister und acht 32-Bit-Fließkomma-Datenregister (*Bild 1*). Kein anderer kommerziell erhältlicher Mikroprozessor verfügt über diese optimale Kombination. Die microNOVA hat beispielsweise zu wenige Register, die microVAX verfügt über zu viele (wie z. B. die CPU 68000). Bei dem Mikrocomputer 68000 findet man keine universellen Register, statt dessen sind einige Register für Daten, die anderen für Adressen vorgesehen. Der Typ 8086 hat keine 32-Bit-Register.

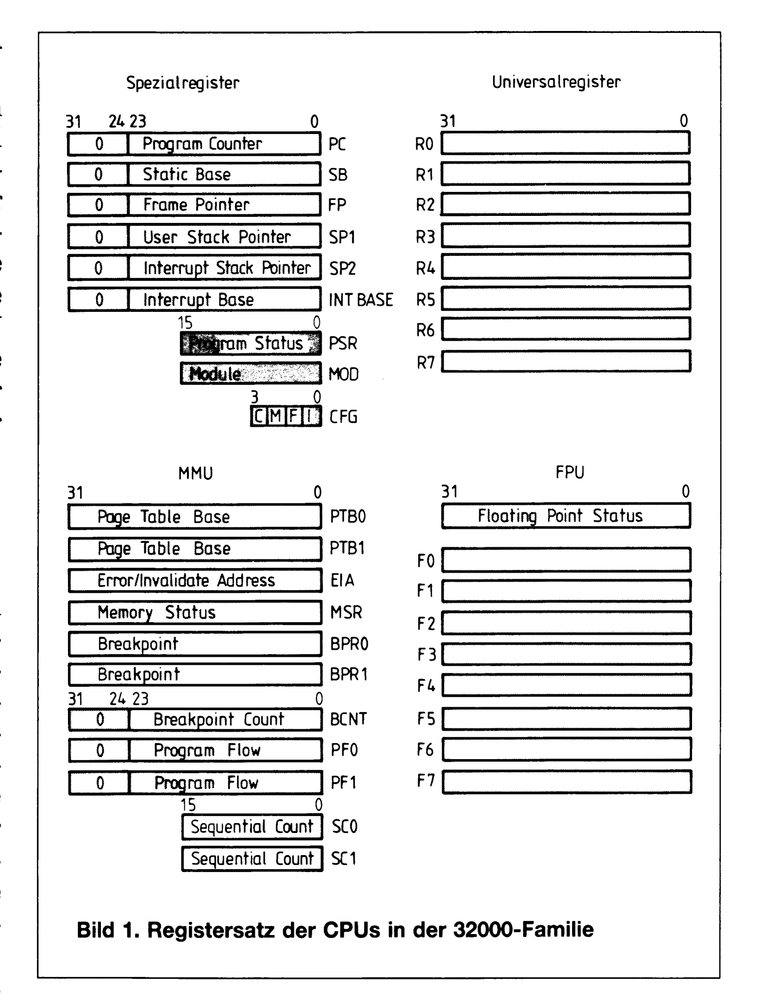


Bild 1. Registersatz der CPUs in der 32000-Familie

Datentypen und Operatoren

Die ersten digitalen Computer wurden ausschließlich für die Lösung von komplexen mathematischen Gleichungen herangezogen. Das sogenannte „Number Crunching" ist auch heute noch bei vielen Computeranwendungen außerordentlich wichtig. In solchen arithmetischen Anwendungen werden vor allen Dingen 32-Bit-Zahlen benutzt, weil Variablen dieser Länge eine höhere Genauigkeit bei der Berechnung bieten. Ein ganzzahliger 32-Bit-Wert entspricht immerhin einer 10stelligen Dezimalzahl. Eine wichtige Forderung an leistungsfähige Mikroprozessoren ist es deshalb, daß ganzzahlige 32-Bit-Werte verarbeitet werden können.

Dieses Merkmal bedeutet, daß der Befehlssatz des Mikroprozessors Operatoren enthält, die direkt (in einer Instruktion) 32 Bit manipulieren können. Das heißt, daß echte 32-Bit-Mikroprozessoren auch 32-Bit-Datentypen in einer einzigen Instruktion manipulieren können müssen. 16-Bit-Mikroprozessoren (wie z. B. der Typ 68000) manipulieren dagegen 16-Bit-Datentypen und benutzen eine Sequenz aus mehreren Instruktionen, um Arithmetik mit 32-Bit-Wortbreiten möglich zu machen. Z. B. erfordert die CPU 68000 drei Befehle zur Multiplikation von zwei 32-Bit-Werten, wobei sich ein 64-Bit-Resultat ergibt. Bei der 32000-Familie genügt eine einzige Instruktion (*Bild 2*).

Neben dem Unterschied zwischen der 32000-Familie und dem Typ 68000 bei der Unterstützung von 32-Bit-Ganzzahlen gibt es einen weiteren grundlegenden Unterschied zwischen den beiden Maschinen bei der Unterstützung von Fließkomma-Zahlen. Die Architektur der 32000-Familie war von Anfang an so konzipiert, daß sie Fließkomma-Arithmetik unterstützt (obwohl der derzeitige technische Stand die Verwendung eines separaten Fließkomma-Prozessors erforderlich macht). Konsequenterweise entschloß man sich für einen Befehlssatz

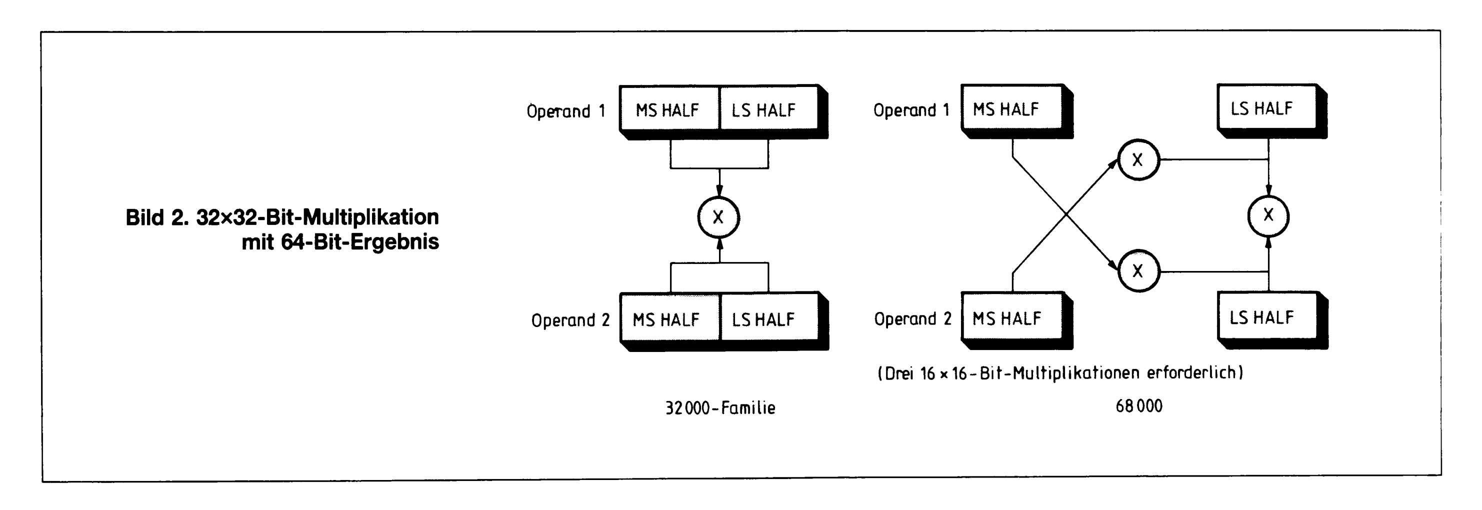


Bild 2. 32×32-Bit-Multiplikation mit 64-Bit-Ergebnis

für arithmetische Operationen mit Fließkomma-Datentypen von 32 Bit und 64 Bit. Diese Befehle sind integraler Bestandteil des gesamten Befehlssatzes. Die *Tabelle* zeigt alle Befehle, die Fließkomma-Operationen unterstützen.

Effizienz und Leistung

Bisher wurden nur zwei prinzipielle Gründe diskutiert, die zur Entwicklung von 32-Bit-Mikroprozessoren führten – der 32-Bit-Adreßbereich und die 32-Bit-Arithmetik. Man darf allerdings nicht den wichtigsten Gesichtspunkt bei der Entwicklung eines Computers aus dem Auge verlieren, nämlich die Leistung. So bot z. B. der Typ iAPX432 einen Adreßbereich von 1 Billion Byte sowie 32-Bit-Arithmetik (tatsächlich 64-Bit-Arithmetik), war aber auf dem Markt nicht erfolgreich, weil seine Leistung nicht ausreichte. Bei der Untersuchung von 32-Bit-Maschinen sollte man immer von realitätsbezogenen Merkmalen ausgehen, z. B. Umfang des Codes und Ausführungsgeschwindigkeit.

Das wichtigste Leistungsmerkmal ist der Code-Umfang. Diese Größe ist der wichtigste begrenzende Faktor für die Arbeitsgeschwindigkeit von Systemen, die auf der Grundlage von Mikroprozessoren arbeiten. Wenn alle anderen Faktoren gleich sind – z. B. Taktgeschwindigkeit, Cache-Kapazität, Busbreite – wird die CPU mit dem kompaktesten Code in der Regel Programme am schnellsten ausführen, weil in einem bestimmten Zeitrahmen die meisten Instruktionen zur ausführenden Einheit gebracht werden können.

Wie beeinflussen die architektonischen Einzelheiten eines Mikroprozessors den Codeumfang und damit die Leistung? Im Prinzip auf zwei Wegen: Erstens kann ein leistungsfähiger Befehlssatz, der über leistungsfähige, symmetrische Adressierarten verfügt, die Anzahl der erforderlichen Befehle zur Codierung eines Algorithmus reduzieren. Zweitens reduzieren die effiziente Codierung dieser Befehle sowie die vielfältige Adressierarten die Anzahl der Bits, die zwischen Speicher und CPU zu transferieren sind. Die Architekten der 32000-Familie nutzten diese beiden Möglichkeiten bei der Konzipierung ihrer Maschine weitgehend aus. Daraus ergibt sich, daß die 32000-Familie über die kompakteste Codedichte aller derzeit erhältlichen Mikroprozessoren verfügt. Bei standardmäßigen Benchmark-Programmen hatte der Code für die 32000-Familie im Mittel 28 % weniger Umfang als derjenige für die CPU 68000, wobei sich bei gleicher Taktfrequenz eine um 46 % höhere Arbeitsgeschwindigkeit ergab (NS32016).

Die leistungsfähigen Befehle, die man bei der 32000-Architektur findet, umfassen auch eine große Zahl, die bei anderen Architekturen mehrere Einzelinstruktionen erfordern. Es gibt Befehle zur Unterstützung von Addition und Subtraktion von gepackten Dezimal-Ganzzahlen. Verschiedene Bit- und Bit-Feld-Operationen unterstützen die Verarbeitung von gepackten Arrays und Records, worauf auf Byte-Grenzen keine Rücksicht genommen werden muß (z. B. kann ein Bit als ein Offset einer Basisadresse an beliebiger Stelle im Speicher adressiert werden). Drei leistungsfähige String-Befehle ermöglichen Zeichenketten-Manipulationen, für die bei anderen Mikroprozessoren Unterprogramme erforderlich sind. Spezielle Array-Index-Berechnungen und Prüf-Befehle werden ebenfalls direkt unterstützt. Es gibt spezielle Instruktionen, die CASE-Statements und FOR-Loops verarbeiten, außerdem unterstützt die Hardware Prozeduraufrufe, was Multitasking und Multiprocessing wesentlich erleichtert.

Die Leistung der 32000-Befehle erhöht sich außerdem durch sehr wirksame Adressierarten, von denen viele bei anderen Mikroprozessoren nicht bekannt sind. Darüber hinaus sind alle 32000-Befehle symmetrisch (das heißt, mit Hilfe jeder Adressierart läßt sich auf alle Operanden zugreifen), so daß Operanden in einem Regi-

Tabelle des Fließkomma-Befehlssatzes

Befehl		Beschreibung
MOVf	gen,gen	Move-Fließkomma
MOVLF	gen,gen	Verschieben und Verkürzen eines Langwertes in einem Kurzwert
MOVFL	gen,gen	Verschieben und Verlängern eines Standardwertes zu einem Langwert
MOVif	gen,gen	Umsetzen von Integerzahlen in Standard- oder Lang-Fließkommawert
ROUNDfi	gen,gen	Umsetzung in Ganzzahlen durch Runden
TRUNCfi	gen,gen	Umsetzung in Ganzzahlen durch Kürzen
FLOORfi	gen,gen	Umsetzen in größten ganzzahligen Wert (größer/kleiner als der Wert)
ADDf	gen,gen	Addieren
SUBf	gen,gen	Subtraktion
MULf	gen,gen	Multiplikation
DIVf	gen,gen	Division
CMPf	gen,gen	Vergleich
NEGf	gen,gen	Invertieren
ABSf	gen,gen	Absolutwert
LFSR	gen	Lade FSR
SFSR	gen	Speichere FSR

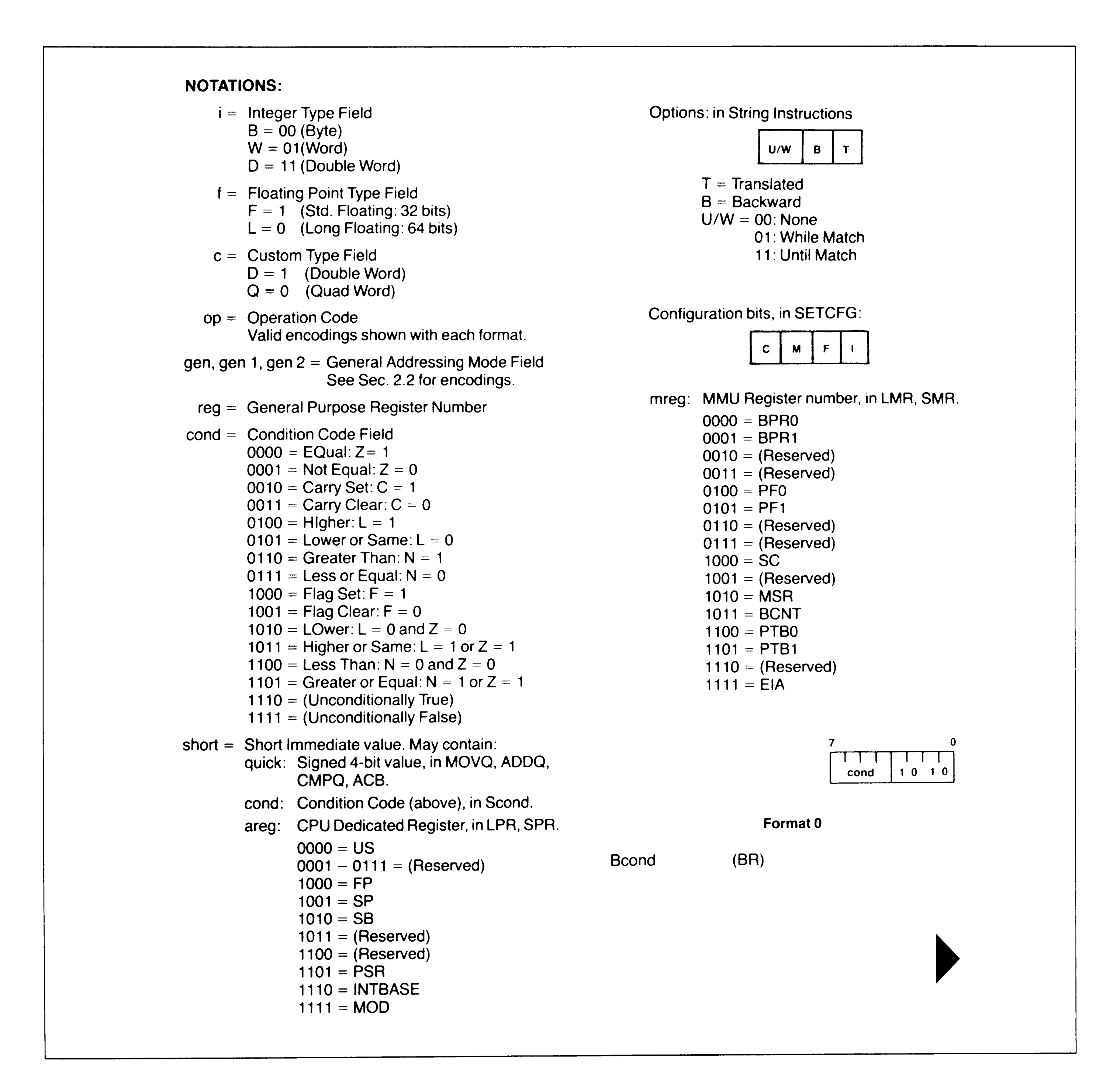
ster, in der Instruktion selbst oder im Speicher sein können. Deshalb können die meisten Arithmetik-Operationen mit einem einzigen Befehl ausgeführt werden. Dagegen ist es beim 68000 nur für den MOVE-Befehl erlaubt, daß sich beide Operanden im Speicher befinden. Daher benötigen alle außer den einfachsten Register-Speicher-Arithmetikoperationen mehrere Instruktionen.

Z. B. erfordert die Operation

$$A = A + B$$

bei der 32000-Familie nur eine einzige Instruktion, unabhängig davon, ob einer der Operanden eine lokale oder globale Variable ist, eine Record-Komponente, ein Array-Element in einem separat kompilierten Modul oder eine beliebige andere Eingabe über eine Hochsprache. Bei den meisten anderen Mikroprozessoren erfordert diese Operation wenigstens drei Instruktionen, wenn die Argumente sich nicht in einem Register befinden.

Die 32000-Familie ist nicht nur so konzipiert, daß sie die Anzahl der Instruktionen zur Implementierung eines Algorithmus reduziert, es werden auch effiziente Codierungstechniken benutzt, um die Anzahl der Bits zur Codierung einer bestimmten Instruktion zu minimieren.



NOTATIONS:

i = Integer Type Field
B = 00 (Byte)
W = 01(Word)
D = 11 (Double Word)

f = Floating Point Type Field
F = 1 (Std. Floating: 32 bits)
L = 0 (Long Floating: 64 bits)

c = Custom Type Field
D = 1 (Double Word)
Q = 0 (Quad Word)

op = Operation Code
Valid encodings shown with each format.

gen, gen 1, gen 2 = General Addressing Mode Field
See Sec. 2.2 for encodings.

reg = General Purpose Register Number

cond = Condition Code Field
0000 = EQual: Z = 1
0001 = Not Equal: Z = 0
0010 = Carry Set: C = 1
0011 = Carry Clear: C = 0
0100 = HIgher: L = 1
0101 = Lower or Same: L = 0
0110 = Greater Than: N = 1
0111 = Less or Equal: N = 0
1000 = Flag Set: F = 1
1001 = Flag Clear: F = 0
1010 = LOwer: L = 0 and Z = 0
1011 = Higher or Same: L = 1 or Z = 1
1100 = Less Than: N = 0 and Z = 0
1101 = Greater or Equal: N = 1 or Z = 1
1110 = (Unconditionally True)
1111 = (Unconditionally False)

short = Short Immediate value. May contain:
quick: Signed 4-bit value, in MOVQ, ADDQ, CMPQ, ACB.
cond: Condition Code (above), in Scond.
areg: CPU Dedicated Register, in LPR, SPR.
0000 = US
0001 – 0111 = (Reserved)
1000 = FP
1001 = SP
1010 = SB
1011 = (Reserved)
1100 = (Reserved)
1101 = PSR
1110 = INTBASE
1111 = MOD

Options: in String Instructions

U/W	B	T

T = Translated
B = Backward
U/W = 00: None
01: While Match
11: Until Match

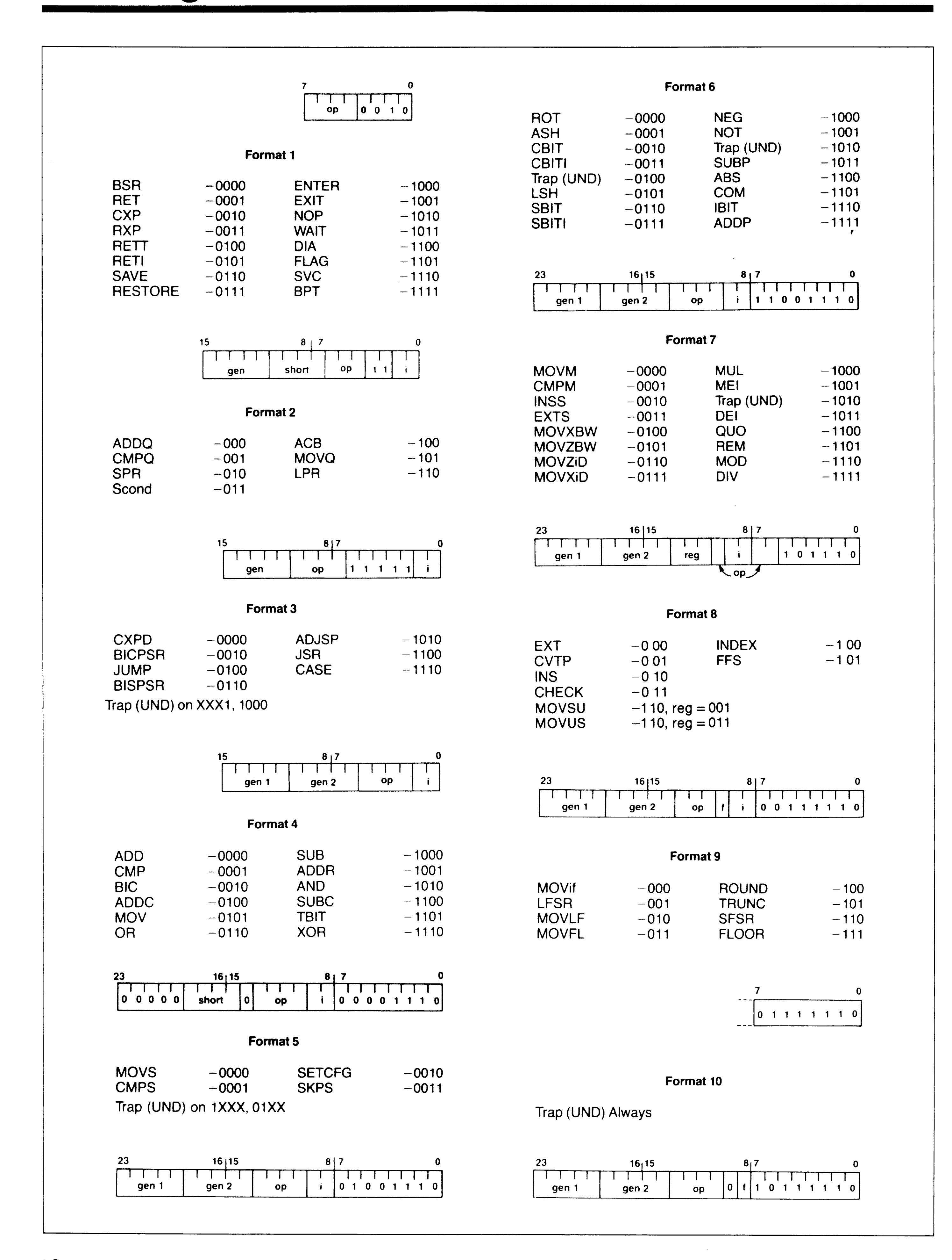
Configuration bits, in SETCFG:

C	M	F	I

mreg: MMU Register number, in LMR, SMR.
0000 = BPR0
0001 = BPR1
0010 = (Reserved)
0011 = (Reserved)
0100 = PF0
0101 = PF1
0110 = (Reserved)
0111 = (Reserved)
1000 = SC
1001 = (Reserved)
1010 = MSR
1011 = BCNT
1100 = PTB0
1101 = PTB1
1110 = (Reserved)
1111 = EIA

Format 0

Bcond (BR)



Format 1

BSR	−0000	ENTER	−1000
RET	−0001	EXIT	−1001
CXP	−0010	NOP	−1010
RXP	−0011	WAIT	−1011
RETT	−0100	DIA	−1100
RETI	−0101	FLAG	−1101
SAVE	−0110	SVC	−1110
RESTORE	−0111	BPT	−1111

Format 2

ADDQ	−000	ACB	−100
CMPQ	−001	MOVQ	−101
SPR	−010	LPR	−110
Scond	−011		

Format 3

CXPD	−0000	ADJSP	−1010
BICPSR	−0010	JSR	−1100
JUMP	−0100	CASE	−1110
BISPSR	−0110		

Trap (UND) on XXX1, 1000

Format 4

ADD	−0000	SUB	−1000
CMP	−0001	ADDR	−1001
BIC	−0010	AND	−1010
ADDC	−0100	SUBC	−1100
MOV	−0101	TBIT	−1101
OR	−0110	XOR	−1110

Format 5

MOVS	−0000	SETCFG	−0010
CMPS	−0001	SKPS	−0011

Trap (UND) on 1XXX, 01XX

Format 6

ROT	−0000	NEG	−1000
ASH	−0001	NOT	−1001
CBIT	−0010	Trap (UND)	−1010
CBITI	−0011	SUBP	−1011
Trap (UND)	−0100	ABS	−1100
LSH	−0101	COM	−1101
SBIT	−0110	IBIT	−1110
SBITI	−0111	ADDP	−1111

Format 7

MOVM	−0000	MUL	−1000
CMPM	−0001	MEI	−1001
INSS	−0010	Trap (UND)	−1010
EXTS	−0011	DEI	−1011
MOVXBW	−0100	QUO	−1100
MOVZBW	−0101	REM	−1101
MOVZiD	−0110	MOD	−1110
MOVXiD	−0111	DIV	−1111

Format 8

EXT	−0 00	INDEX	−1 00
CVTP	−0 01	FFS	−1 01
INS	−0 10		
CHECK	−0 11		
MOVSU	−1 10, reg = 001		
MOVUS	−1 10, reg = 011		

Format 9

MOVif	−000	ROUND	−100
LFSR	−001	TRUNC	−101
MOVLF	−010	SFSR	−110
MOVFL	−011	FLOOR	−111

Format 10

Trap (UND) Always

Format 11

ADDf	-0000	DIVf	-1000
MOVf	-0001	Trap (UND)	-1010
CMPf	-0010	Trap (UND)	-1011
SUBf	-0100	MULf	-1100
NEGf	-0101	ABSf	-1101
Trap (UND)	-0110	Trap (UND)	-1110
Trap (UND)	-0111	Trap (UND)	-1111

7 0
1 1 1 1 1 1 1 0

Format 12

Trap (UND) Always

7 0
1 0 0 1 1 1 1 0

Format 13

Trap (UND) Always

23 16 15 8 7 0
gen 1 | short | 0 | op | i | 0 0 0 1 1 1 1 0

Format 14

RDVAL	-0000	LMR	-0010
WRVAL	-0001	SMR	-0011

Trap (UND) on 01XX, 1XXX

23 16 15 8 7 0
n n n 1 0 1 1 0

Operation Word — ID Byte

Format 15
(Custom Slave)

nnn — Operation Word Format

000: 23 16 15 8 — gen 1 | short | x | op | i

Format 15.0

CATST0	-0000	LCR	-0010
CATST1	-0001	SCR	-0011

Trap (UND) on all others

001: 23 16 15 8 — gen 1 | gen 2 | op | c | i

Format 15.1

CCV3	-000	CCV2	-100
LCSR	-001	CCV1	-101
CCV5	-010	SCSR	-110
CCV4	-011	CCV0	-111

101: 23 16 15 8 — gen 1 | gen 2 | op | x | c

Format 15.5

CCAL0	-0000	CCAL3	-1000
CMOV0	-0001	Trap (UND)	-1010
CCMP	-0010	Trap (UND)	-1011
CCAL1	-0100	CCAL2	-1100
CMOV2	-0101	CMOV1	-1101
Trap (UND)	-0110	Trap (UND)	-1110
Trap (UND)	-0111	Trap (UND)	-1111

If nnn = 010, 011, 100, 110, 111
then Trap (UND) Always

7 0
0 1 0 1 1 1 1 0

Format 16

Trap (UND) Always

7 0
1 1 0 1 1 1 1 0

Format 17

Trap (UND) Always

7 0
1 0 0 0 1 1 1 0

Format 18

Trap (UND) Always

7 0
X X X 0 0 1 1 0

Format 19

Trap (UND) Always

Implied Immediate Encodings:

7 0
r7 r6 r5 r4 r3 r2 r1 r0

Register Mask, appended to SAVE, ENTER

7 0
r0 r1 r2 r3 r4 r5 r6 r7

Register Mask, appended to RESTORE, EXIT

7 0
offset | length – 1

Offset/Length Modifier appended to INSS, EXTS

Bild 3. Befehlsformate (Auszug aus Original-Datendokumentation)

Die Operationscodes der 32000-Familie wurden sorgfältig den Befehlen zugeordnet, so daß häufig benutzte Instruktionen sehr kurze Codes haben, während relativ selten benutzte, aber außerordentlich leistungsfähige Befehle mit längeren Opcodes arbeiten.

Die Codierung von Displacements in Instruktionen wurde auch so vorgenommen, daß lokale Referenzen sehr kompakt sind und daß sich keine Nachteile beim Zugriff auf etwas weiter entfernte Daten ergeben. Displacements können 1, 2 oder 4 Byte lang sein – nur so lang wie notwendig, aber trotzdem in der Lage, auf den gesamten Adreßbereich zuzugreifen. Beim Typ 68000 findet man hier ein 16-Bit-Displacement, das die Indizierung auf einen Bereich von 64 KByte beschränkt. Weil Displacements in einem Programm typischerweise klein sind, lassen sich diese in einem einzigen Byte codieren. Damit ist die relative Adressierung des Stacks, von Frame- und statischen Basisregistern sehr effizient.

Spezielle schnelle Befehle ermöglichen die sehr kompakte Codierung von häufig benutzten Instruktionen. Die sogenannten „Quick-Formen" für Addition, Verschiebung und Vergleich codieren einen kleinen ganzzahligen Operanden (im Bereich −8...+7) anstelle einer zweiten allgemeinen Adressierart. So erfordert beispielsweise die Compare-Quick-Instruktion beim 32000, die einen Immediate-Wert mit einem langen Wert in einem Register vergleicht, lediglich drei Taktzyklen (300 ns bei 10 MHz). Beim 68000 erfordert diese gleiche Operation 6 Bytes und 14 Taktzyklen (1400 ns bei 10 MHz).

Effiziente Codierung sowohl von Opcode als auch Displacement in Befehlen der 32000-Familie führt zu Befehlslängen, die zwischen einem und 25 Bytes liegen, abhängig von der Häufigkeit der Benutzung sowie der Komplexität der Adressierart. Darüber hinaus sind fast alle Immediate-Befehlsgrößen zu finden. Der 32000-Befehlssatz ist so konzipiert, daß er jeden Befehl mit einer minimalen Anzahl von Bits codiert (*Bild 3*). Die Befehle verlängern sich in Schritten von einem Byte. Beim 68000 müssen die Befehle eine Länge des Mehrfachen von 16 Bit betragen.

Eine weitere Steigerung der Codierungseffizienz ergab sich bei der 32000-Familie dadurch, daß man es erlaubte, daß Befehle und Daten (unabhängig von dem Datentyp) für jede Byte-Grenze normiert werden können. Man muß sich hier nicht an die Wortgrenzen (16 Bit) halten. Wenn diese Normierung mit geradzahligen Adressen übereinstimmen muß (wie beim 68000), vereinfacht das natürlich die Implementierung der Architektur, allerdings nur auf Kosten einer höheren Schwierigkeit für den Programmierer (und den Compiler) sowie der geringeren Effizienz und Speicherausnutzung bei Daten- und Befehls-Speicherung. Wenn beispielsweise ein Record zwei Elemente enthalten soll, von denen eines die Größe eines Bytes und das andere die Größe eines Wortes hat, sind bei der 32000-Familie lediglich drei Byte im Speicher erforderlich. Bei einer Maschine, die mit geradzahliger Adreßnormierung arbeitet, müssen hierfür 4 Byte vorgesehen werden. Dies ist im Einzelfall vielleicht eine kleine Speicherkapazität, aber für ein Array von 2000 solcher Records fallen auf diese Weise bereits 2000 zusätzliche unbenutzte Bytes an.

Um die Ausführungsgeschwindigkeit weiterhin zu erhöhen, implementiert die 32000-Familie einen sogenannten Instruction-Look-Ahead-Mechanismus, der den Befehlsstrom in eine 8-Byte-Warteschlange aufteilt und aufnimmt. Eine 3stufige Ausführungs-Pipeline erlaubt es, daß drei aufeinanderfolgende Instruktionen gleichzeitig ausgeführt werden: Während eine Instruktion ausgeführt wird, arbeitet der Prozessor bereits an der Decodierung der folgenden und die nächste wird bereits durch die Warteschlange geschoben.

Die meisten Merkmale der 32000-Familie, die in diesem Abschnitt beschrieben werden, erfordern zusätzliche Hardware-Mechanismen für ihre Implementierung. Z. B. sind leistungsfähige Befehle und symmetrische Adressierarten wesentlich schwieriger zu implementieren als einfache Befehle und Spezial-Adressierarten. Byte-Normierung ist schwieriger zu implementieren als Wort-Normierung. Auch der Prefetch-Mechanismus (Instruction-Look-Ahead) macht die Implementierung ein wenig komplizierter. Die Entwickler der 32000-Familie waren aber der Meinung, daß es wichtig ist, Aufwand und Zeit in diese zusätzlichen Mechanismen zu investieren, um die höchstmögliche Leistung für ihre Maschine zu erreichen. In der Praxis hat sich erwiesen, daß dieser Weg richtig war.

Kompatibilität und Familienkonzept

Im Jahre 1964 kam die Firma IBM auf die revolutionierende Idee, mit dem System 360 erstmals eine Computerfamilie auf den Markt zu bringen. Bis zur Entwicklung dieses Konzeptes waren immer dann, wenn neue Maschinen auf den Markt kamen, alle für frühere Maschinen dieses Herstellers geschriebenen Programme unbrauchbar. Man mußte alle Programme für die jüngere Maschine neu schreiben, weil sich wichtige Unterscheidungen in der Architektur ergaben. Diese unglaubliche Verschwendung von Geld und Zeit führte dazu, daß man eine einzige Architektur bei Maschinen unterschiedlicher Leistungsfähigkeit verwendete. Programme, die für weniger leistungsfähige Maschinen geschrieben wurden, liefen auch auf größeren Maschinen. Wesentlich wichtiger war allerdings, daß IBM garantieren konnte, daß jede spätere Maschine auch über die gleiche Architektur verfügte. Dieses Versprechen hat IBM bisher eingehalten, so daß auch heute übliche Großrechner Programme verarbeiten können, die Anfang der 60er Jahre geschrieben wurden.

Das Familienkonzept, das von IBM einmal begonnen wurde, fand man bald auch bei Minicomputern. Die PDP-11- und VAX-11-Familie von DEC sowie die NOVA/ECLIPSE-Familie von Data General sind dafür Beispiele. Der allgemeine Ausdruck für dieses Familienkonzept ist „Aufwärts-Kompatibilität". Damit ist

gemeint, daß neuere, leistungsfähigere Maschinen so konzipiert sind, daß sie auch Programme für die älteren, weniger leistungsfähigeren Maschinen verarbeiten können. Der umgekehrte Weg ist allerdings nicht immer möglich, denn neue Maschinen bieten sehr häufig zusätzliche Merkmale, die bei den älteren Versionen nicht zu finden sind. Programme, die keinen Gebrauch von diesen Programmen machen, lassen sich auch auf den älteren Maschinen benutzen.

Bei Mikroprozessoren wurde in der Vergangenheit das Familienkonzept und die Aufwärts-Kompatibilität nicht so häufig benutzt. Die Architektur des Intel-Prozessors 8080 wurde zu einem solchen Standard in der 8-Bit-Welt, daß sowohl der Typ Z80 von Zilog sowie die spätere Intel-CPU 8085 so konzipiert waren, daß sie 8080-Software verarbeiteten (das heißt, daß sie aufwärtskompatibel zum 8080 waren). Damit lassen sie sich durchaus als einer 8080-Familie zugehörig bezeichnen, obwohl man zunächst nicht an die Entwicklung einer solchen Familie gedacht hatte. In der 16-Bit-Welt versuchten sowohl Intel als auch Motorola, eine architektonische Kompatibilität in ihren Produktlinien einzuhalten, wobei sie dies allerdings mit unterschiedlichem Erfolg realisieren konnten. Die Intel-Typen 8088 und 8086 sowie die Motorola-Typen 68000 und 68008 waren wohl am erfolgreichsten. In beiden Fällen wurde eine 16-Bit-Architektur mit 16-Bit-Implementierung (8086 und 68000) von einer Implementierung der gleichen Architektur, aber mit einem externen 8-Bit-Bus (68008 und 8088) gefolgt. Diese beiden Paare von Maschinen können genau die gleiche Software verarbeiten, so daß sie beide aufwärts- und abwärts-kompatibel sind.

Andererseits waren Intel und Motorola wesentlich weniger erfolgreich bei der Einführung von leistungsfähigeren Mitgliedern der 16-Bit-Familie. Der Typ 68010 von Motorola hat beispielsweise ein völlig anderes Stack-Format als der Typ 68000, was bedeutet, daß er inkompatibel zur früher auf den Markt gekommenen Maschine ist. Obwohl der Typ 286 von Intel eine spezielle Kompatibilitäts-Betriebsart besitzt, ist diese CPU völlig inkompatibel zum 8086 in der Native-Betriebsart. Der Grund für diese Inkompatibilität ist, daß die ursprünglichen Architekten sowohl der CPU 8086 als auch 68000 ihre Maschinen nicht unter dem Gesichtspunkt der zukünftigen Erweiterung konzipierten. Man dachte damals nur an die zur Verfügung stehenden Technologien und sah keinen Grund, bereits 32-Bit-Merkmale zu berücksichtigen, die sich damals noch nicht direkt implementieren ließen. Konsequenterweise wurden Memory-Mapping, virtuelle Speicher, Fließkomma- und andere 32-Bit-Operationen bei der ursprünglichen architektonischen Konzipierung vollständig ignoriert, was zu den heutigen Problemen mit der Inkompatibilität führte.

Es soll hier noch einmal darauf hingewiesen werden, daß Aufwärtskompatibilität sehr wichtig ist. IBM, DEC und Data General haben sich in den vergangenen Jahren gerade damit sehr viel Mühe gegeben. Der Grund: Die Kunden haben einen echten Bedarf dafür! Diese müssen

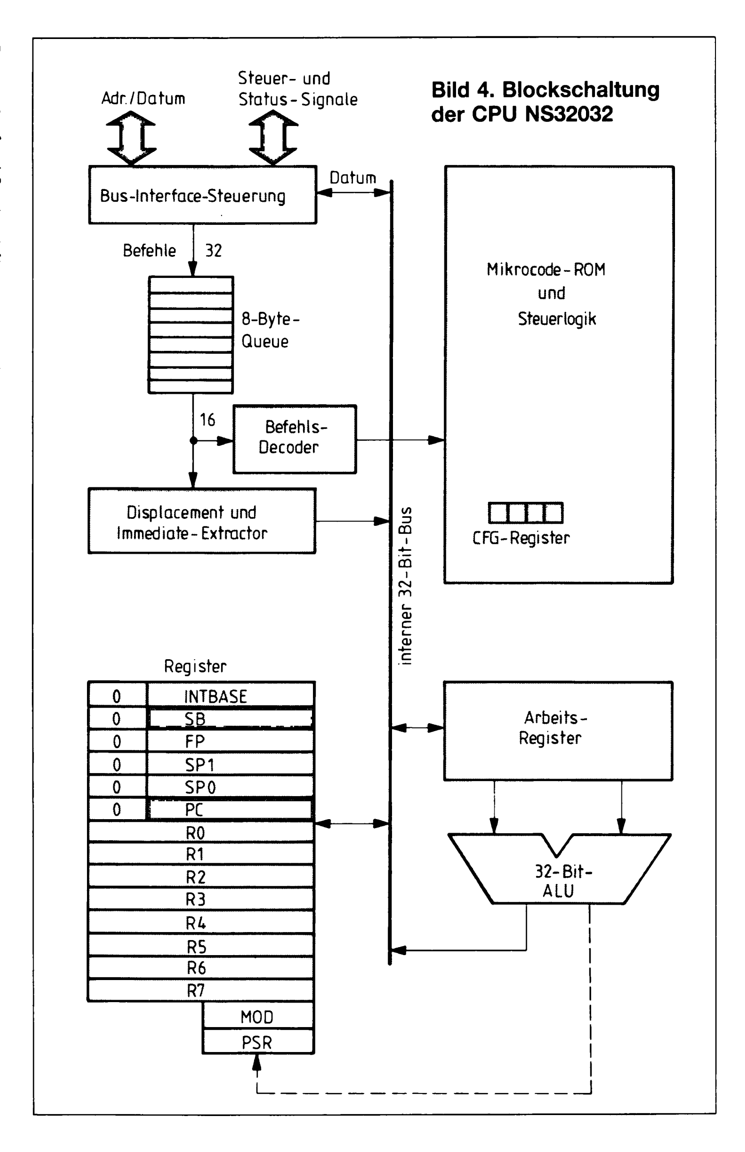


Bild 4. Blockschaltung der CPU NS32032

nämlich wesentlich mehr in ihre Software investieren als in die Hardware, und sie können unmöglich alle ihre Programme neu schreiben. Möglicherweise haben Halbleiterfirmen andere Erfahrungen als Computerhersteller. Man erkannte vielleicht nicht die Probleme der Kunden mit der Inkompatibilität einer Produktlinie. National Semiconductor achtete bei der Konzipierung der 32000-Familie von Anfang an auf strikte Einhaltung der Kompatibilität. Man wollte schon zu Beginn eine Architektur entwickeln, die über mehrere Jahrzehnte akzeptabel ist, auch wenn die Technologie sich ändert und viele andere Implementierungen auf den Markt kommen.

Der wichtigste Aspekt der 32000-Familie ist, daß alle Mitglieder nicht nur aufwärts-kompatibel sind. Sie sollen aufwärts- und abwärts-kompatibel sein. Das bedeutet, daß sie vollständig kompatibel sind und sich lediglich in Details unterscheiden, die keinen Einfluß auf die architektonische Gestaltung haben. Programme, die für ein beliebiges Mitglied dieser Familie geschrieben sind, laufen auch auf allen anderen Familienmitgliedern. Das bedeutet, daß die für eine Maschine zur Verfügung stehende Software sehr umfangreich ist im Vergleich zu

Maschinen, für die nur Aufwärts-Kompatibilität garantiert wird. Wenn man die enormen Kosten für die heutige Softwareentwicklung in Betracht zieht, kann man erkennen, welchen Stellenwert diese Tatsache einnimmt.

Die Entwickler der 32000-Familie konnten eine vollständig kompatible Architektur konzipieren, indem sie über die Grenzen der derzeit verfügbaren Technologie herausschauten und für die Zukunft entwickelten. Obwohl Fließkomma-Arithmetik und virtuelle Speicher sich mit einer CPU heute noch nicht auf einem Chip zusammenfassen lassen, wurden architektonische Merkmale realisiert, die dies eines Tages zulassen. Heute sind noch separate Fließkomma- und Speicherverwaltungs-Einheiten erforderlich, eines Tages werden sie sich aber bestimmt auf einem Chip mit der CPU befinden. Wenn dieser Integrationsschritt einmal möglich ist, wird damit die gleiche Architektur implementiert, wie das bei den heutigen Versionen auch der Fall ist.

Die einzigen Unterschiede, die sich bei den verschiedenen Versionen der 32000-Familienmitglieder ergeben, sind folgende:

32-Bit-Prozessoren etablieren sich

Der technische Fortschritt auf dem Mikroprozessor-Sektor ist gekennzeichnet durch die Entwicklung in folgenden vier Bereichen: CMOS-Technologie, 32-Bit-Prozessoren, neue leistungsfähige 16-Bit-Mikrocontroller und wachsende Verzahnung von Hard- und Software. Die Leistungsfähigkeit der Systemarchitektur eines Prozessors hat zunehmende Bedeutung, da die Anwender mehr und mehr eine Entwicklungsbasis fordern, die auf lange Sicht klare Ausbaumöglichkeiten gewährleistet.

Die Strategie muß langfristig angelegt sein

Die vier genannten Aspekte sind untrennbar miteinander verknüpft. Zum Beispiel ist die Entwicklung zukünftiger Hochleistungs-Prozessoren in 32-Bit-Technik von der Verfügbarkeit zuverlässiger, reproduzierbarer CMOS-Fertigungsverfahren abhängig. Zwei von drei neuen 16-Bit-Mikrocontrollern werden heute in CMOS-Technologie realisiert, was einen entscheidenden Einfluß auf die Leistungsfähigkeit dieser Bauelemente hat.

Es ist weiterhin eine Tatsache, daß kaum ein Anwender sich für einen noch so leistungsfähigen Prozessor interessieren dürfte, für den es keine Unix-Portierung gibt. Von gleichrangiger Bedeutung sind Compiler für die wichtigsten Programmiersprachen.

Schon immer war die Halbleiterindustrie die schnellebigste der technisch orientierten Industriesparten. Investitionen, technische Weiterentwicklungen und kommerzielles Wachstum waren die drei Schlüsselfaktoren für sicheren Erfolg. In einem solchen Umfeld rächen sich Fehleinschätzungen, jedoch bietet sich andererseits auch eine gute Chance zur Neuorganisation und zum Beheben von Fehlern.

Die hier angesprochenen Gesichtspunkte beziehen sich auf Entwicklungen sehr hoher Komplexität, die ihrerseits wiederum nur Elemente einer globalen Strategie sind. Längere Produkt-Entwicklungszyklen erhöhen die Bedeutung richtiger Grundsatzentscheidungen, sind doch grundsätzliche konzeptionelle Fehler schwierig oder gar unmöglich wiedergutzumachen.

Die technischen Entwicklungen sollten deshalb vor ihrem jeweiligen kommerziellen Hintergrund gesehen werden. Das Marktmodell von National Semiconductor beispielsweise geht von überdurchschnittlichen Zuwachsraten auf dem Gebiet der 32-Bit-Prozessoren und der 16-Bit-Mikrocontroller aus. Weil sich diese Zuwachsraten aus einer sehr kleinen Basis heraus entwickeln, stellen sie kurzfristig keinen bedeutenden Marktsektor dar, bilden aber die Richtschnur für die zukünftige Entwicklung.

Der 32-Bit-Prozessor im 8-Bit-System?

Wie eindrucksvoll diese Aussichten auch sein mögen, so ist *Hans Rohrer*, europäischer Marketing-Manager bei National Semiconductor doch der Ansicht, daß in dieser Betrachtungsweise wichtige dynamische Faktoren unberücksichtigt bleiben, die die Wachstumstendenzen beeinflussen. Rohrer meint, daß die Umstellung vieler Hersteller auf 32-Bit-Systeme unter Beibehaltung der bestehenden Systemkonfigurationen den Fortschritt auf dem 32-Bit-Sektor bedeutend vorantreiben wird, und zwar auf Kosten älterer, bereits eingeführter Produkte.

Er vertritt die Auffassung, daß das Entwicklungsmanagement folgenden Leitlinien folgen wird:

a) Nutzung der unmittelbaren Vorteile einer leistungsfähigen 32-Bit-Architektur.
b) Einführung eines Mikroprozessor-Standards, der den Erfordernissen der nächsten 10 bis 15 Jahre entspricht.

Dieser Standpunkt bedingt ein Umdenken hinsichtlich der klassischen Definition von 4-, 8-, 16- und 32-Bit-Systemen, um das scheinbare Paradoxon eines 32-Bit-Prozessors in einem 8-Bit-System zu erklären.

Üblicherweise bezog sich die Bit-Angabe auf die Breite des Datenbusses, auf den die übrige Architektur eines Systems ausgelegt war. Dies jedoch impliziert eine Vereinfachung, die bei der Charakterisierung der neuesten Bauelemente-Generation nicht zulässig ist.

Ein Beispiel für diese Inkonsequenz wären ein 4-Bit-Mikrocontroller mit einem 8-Bit-Befehlsspeicher, der mit 11-Bit-breiten Adressen arbeitet, oder eine 8-Bit-CPU mit teilweiser 16-Bit-Datenverarbeitung und der Fähigkeit zur Berechnung von 16-Bit-Adressen.

Der abgestimmte Datenbus

32-Bit-Produktfamilien ermöglichen eine freie, an der Anwendung orientierte Festlegung des externen Datenbusses. Der Systementwickler hat also die Möglichkeit, den Datenbus exakt auf die Anforderungen seines Systems abzustimmen, ohne deswegen gezwungen zu sein, eine CPU mit geringerer Verarbeitungsleistung einzusetzen. Außerdem besteht so die Gewißheit, bei späteren Entwicklungen einen abweichenden Datenbus verwenden zu können, ohne daß bisher geschriebene Software wertlos wird. Sämtliche CPUs

– Technologische Merkmale, z. B. Strukturgröße, Anzahl der Gatterstrukturen auf einem Chip, Anzahl der Chips zur Implementierung der Architektur.
– Breite der externen Busse.
– Anzahl der ALU, das heißt, Anzahl der Berechnungen, die parallel auszuführen sind.

Die 32-Bit-Architektur der 32000-Familie wird derzeit in drei VLSI-Schaltungen implementiert: eine CPU, ein Fließkommaprozessor und eine Speicherverwaltungs-Einheit. Darüber hinaus gibt es vier Versionen der CPU: die Typen NS32008, NS32016, NS32032 und NS32332. Jede hat die gleiche 32-Bit-Architektur mit der vollständigen internen 32-Bit-Implementierung. Unterschiede gibt es nur bei der Breite der Datenpfade zum Speicher: 8 Bit, 16 Bit oder 32 Bit. *Bild 4* zeigt als Beispiel die Struktur des Prozessors 32032. Diese hat intern und extern 32-Bit-Busse sowie eine 32-Bit-Architektur. Bei typischen Anwendungen benutzt diese CPU lediglich 50 % der verfügbaren Bus-Bandbreite, so daß sie sich für komplexe Systeme eignet, bei denen mehrere Zentralprozessoren, DMA-Einheiten und Grafik-Verarbeitung mit hoher Geschwindigkeit erforderlich sind.

einer Serie wären nämlich (zumindest im Idealfall) softwarekompatibel.

Der Kampf um die Führungsposition

Der Kampf um die Spitzenposition auf dem 32-Bit-Sektor setzte in vollem Umfang ein, als die ersten Exemplare des MC68020 ausgeliefert wurden. Dieser Prozessor ist der erste Konkurrent des Typs NS32032 von National Semiconductor, der in Produktionsstückzahlen gefertigt wird. Die Firmen Zilog und Intel präsentieren ebenfalls 32-Bit-Prozessoren, den Z80000 bzw. den 80386.

Der „Transputer" von Inmos stellt eine radikale Abkehr von den traditionellen Mikroprozessor-Architekturen dar. Diese grundlegend neue Eigenschaft könnte zur Folge haben, daß das Produkt weder große Verbreitung, noch ernstzunehmenden kommerziellen Einfluß bekommt. Es mag jedoch spezielle Anwendungsfälle geben, in denen die besonderen Eigenschaften des Transputers Vorzüge aufweisen, die von konventionellen Systemen nicht geboten werden. In diesen Applikationen muß intensive Integrationsarbeit geleistet werden.

Das Hauptproblem für den Transputer-Hersteller besteht darin, auf der Basis einer sehr kleinen und hochspezialisierten Anwendergemeinde die Entwicklung und Unterstützung voranzutreiben, während parallel dazu versucht werden muß, die „Applikationslücke" zu schließen, um dem Transputer eine breitere Akzeptanz zu eröffnen. Die Wahrscheinlichkeit ist groß, daß hierfür ein Partner notwendig ist, um die Kosten zu teilen und die verfügbaren Ressourcen zu vergrößern.

Man wird nach Wegen suchen, die enggekoppelte Mehrprozessor-Architektur an wichtige Applikationen anzupassen. Viele Diskussionen kreisen um die Vor- und Nachteile der Konzeption des Transputers, die der Hersteller als „RISC-Architektur" (Reduced Instruction Set Computer = Computer mit reduziertem Befehlssatz) bezeichnet.

CMOS sichert den Erfolg

Bereits Ende der 70er Jahre war sich die Mehrheit der Elektronikindustrie darin einig, daß der langfristige Erfolg der Mikroprozessoren von der CMOS-Technologie abhängen würde. Die Begründung hierfür war einfach: Im Verlauf der 80er und 90er Jahre würde die Fertigungstechnik Chips mit mindestens 1 Million Transistoren ermöglichen, die in NMOS-Technik mehr Wärme entwickeln, als mit den bestehenden Gehäusetechniken abzuleiten ist.

Ein Forschungsprogramm zur Verbesserung der Schaltgeschwindigkeit von CMOS-Chips und zur Erhöhung der Pakkungsdichte hat gute Ergebnisse gebracht, sind doch Produkte mit mehr als 200 000 Transistoren und einer Taktfrequenz von 17 MHz realisierbar.

Mäßige Leistungsfähigkeit

Von jeher werden Mikrocontroller als äußerst preisgünstige, maskenprogrammierbare, spezielle Massenprodukte angesehen, deren Leistung jedoch eher mäßig ist. Dieses Image hat sich durch neue Produkte entscheidend gewandelt.

Die Typen 8096 von Intel, 68HC11 von Motorola und HPC1600 von National ermöglichen Taktfrequenzen bis zu 16 MHz, 16-Bit-Verarbeitung und einen großen integrierten RAM-Speicher von 256 Byte. Außerdem werden integrierte maskenprogrammierbare Festwertspeicher mit großer Kapazität angeboten. Die erhöhte Packungsdichte gibt zusätzliche Anschlüsse für E/A-Funktionen frei.

Jake Nelson, aus der Applikationsabteilung von Intel, wies darauf hin, daß der 8096 ursprünglich für Motorsteuerungen in Kraftfahrzeugen konzipiert wurde. Er ist in der Lage, Abläufe zu bearbeiten und auszulösen, während parallel dazu das Hauptprogramm läuft.

Eine interessante Eigenschaft dieser Chips besteht in der Möglichkeit, auf äußerst flexible Weise optionale Subsysteme in den Chip zu integrieren (z. B. serielle E/A-Funktionen, A/D- und D/A-Umsetzer). Dies ist ein bedeutender Schritt in Richtung auf das voll anwenderprogrammierbare System auf einem Chip.

Die Hardware-Eigenschaften, die Geschwindigkeit und die Verarbeitungsleistung dieser Komponenten legen gemeinsam mit dem relativ niedrigen Preis den Schluß nahe, daß zahlreiche 8-Bit-Applikationen mit diesem Bauelement gut bestückt wären.

1985 war das Jahr, in dem das 32-Bit-Zeitalter richtig begann. Die deutlich leistungsverbesserten Architekturen werden sich in allen Anwendungen positiv auswirken. Die hervorgebrachten Normen werden darüber hinaus bis zum Ende des Jahrhunderts Gültigkeit besitzen.

Bei der Auswahl eines geeigneten Produkts werden die Anwender den gleichen Entscheidungsprozeß durchmachen, den die Entwickler dieser neuen Bauelemente vor Jahren absolvierten, als es die neue Produktgeneration zu konzipieren galt. Jeder Anwender sollte sich über die immense Bedeutung im klaren sein, die diese Entscheidung für den zukünftigen Geschäftserfolg hat. Anthony Amend

Günther Hausmann

Die beste Unix-Maschine

Seitdem Unix-Systeme, ursprünglich im Bereich der Minicomputer zu Hause, auch bei den kleinen Rechnern Fuß faßten, gibt es tiefgreifende Diskussionen darüber, welcher Mikroprozessor die besten Umgebungsbedingungen für dieses Betriebssystem mit sich bringt. Während 16-, 16/32- und 32-Bit-Mikroprozessoren der jetzigen Generation bereits die erforderlichen großen Adreßbereiche und leistungsfähigen Befehlssätze bieten, besitzt die Serie 32000 besonders interessante Merkmale, die die Implementierung von Unix und Hochsprachen erleichtern. Dieser Beitrag soll solche Punkte herausarbeiten, z. B. die Eignung des Befehlssatzes für die Hochsprachenunterstützung, den virtuellen Speicher mit dem Demand-Paged-Zugriff und die Auswirkungen der Fließkomma-Hardware auf die Leistungsfähigkeit des Gesamtsystems. Als Beispiel für ein Betriebssystem, das auf der Serie 32000 implementiert ist, zeigt dieser Beitrag abschließend GENIX, eine Implementierung von Unix, Version Berkeley 4.1.

Der richtige Befehlssatz

Der Grundbefehlssatz für die CPUs aus der 32000-Serie unterstützt nicht nur Hochsprachenoperationen, sondern auch Datentypen auf höherer Ebene. Weil darüber hinaus Grunddatentypen, wie z. B. Integer, Pointer, BCD-Integer, Boolesche Variablen, Bit- und Fließkomma-Formate, unterstützt werden, gibt es bei dem Befehlssatz der Serie 32000 keine Probleme mit Datenstrukturen wie z. B. Arrays, Records, gepackten Strukturen (Arrays und Records) sowie Strings.

Die Architektur der Serie 32000 arbeitet mit neuen Adressierarten, vier davon sind speziell für Hochsprachen zugeschnitten: relativ, extern, skalierte Indizierung sowie Top-Off-Stack. Relative Adressierung ermöglicht, daß der Pointer sich in einem Speicherplatz und nicht in einem Register befindet. Externe Adressierung unterstützt das Software-Modul-Konzept der Serie 32000, bei

```
static int oof;
static struct two_words { int one, two;} *twop;
extern int ext;

main()
{
  int local;
  struct two_words *local_twop;
  register regstr;
  register struct two_words *regstr_twop;

  regstr = 0;               /* direct register addressing (movdq 0,r7)               */
  regstr_twop- > two = 0;   /* offset register addressing (movdq 0,4(r6))            */
  local = 0;                /* frame pointer relative (movdq 0,-4(fp))               */
  local_twop- > two = 0;    /* frame pointer memory relative (movdq 0,4(-8(fp)))     */
  oof = 0;                  /* static memory relative (movdq 0,_oof)                 */
  twop- > two = 0;          /* static memory relative (movdq 0,4(r0))                */
  ext = 0;                  /* external address mode (movdq 0,_ext)                  */

  *((int*)1234) = 0;        /* absolute address mode (movdq 0,@1234)                 */
}
```

Bild 1. Externe Adressierung unterstützt das Software-Modul-Konzept der Serie 32000. Bei diesem Programm ist die Beziehung zwischen der Unix-Programmiersprache C und den Adressierarten zu erkennen

```
static int oof[10];
static struct two_words { int one[10], two[10]; } *twop;
extern ext[10];
main()
{
  int local[10];
  struct two_word *local_twop;
  register regstr;
  register struct two_words *regstr_twop;

  regstr_twop->two[regstr]=0;     /* offset register addressing (movqd 0,0(r0)[r7:d[()          */
  local[regstr]=0;                /* frame pointer relative (movqd 0,40(fp)[r7:d])              */
  local_twop->two[regstr]=0;      /* frame pointer memory relative (movqd 0,4(-8(fp)))          */
  oof[regstr]=0;                  /* static memory relative (movqd 0,_oof[r7:d])                */
  twop->two[regstr]=0;            /* static memory relative (movqd 0,0(r0)[r7:d])               */
  ext[regstr]=0;                  /* external address mode (movqd 0,_ext[r7:d]                  */

  *((int *) 1234+regstr)=0;       /* absolute address mode (movqd 0,1234[r0])                   */
}
```

Bild 2. Programm wie im Bild 1 mit indizierter Adressierungsart

dem es möglich ist, daß Module ohne Linkage-Editierung relociert werden. Das in *Bild 1* gezeigte Programm läßt die Beziehungen zwischen der Unix-Programmiersprache C und diesen Adressierarten erkennen. An der entsprechenden Stelle identifiziert ein Kommentar die Adressierart, die von jedem Statement generiert wurde.

Skalierte Indizierung läßt sich jeder Adressierart hinzufügen, um die Adresse mit einem Index der Größe 1, 2, 4 oder 8 auszustatten, wodurch sich ein einfacher Weg eröffnet, Adreß-Arrays im Byte-, Wort-, Doppelwort- oder Vierfachwort-Format zu bilden. *Bild 2* zeigt das gleiche Programm wie in Bild 1 mit dem Unterschied, daß hier eine indizierte Adressierart benutzt wurde. Jede Zeile erzeugt eine bestimmte Adressierart.

Die Top-Off-Stack-Adressierung erlaubt für alle Befehle die Manipulation oder den Bezug auf einen Operanden im Stack. Diese Betriebsart eignet sich insbesondere für stack-orientierte Systeme, wie sie bei Unix typisch sind und bei denen die Compiler die Top-Stack-Betriebsart dazu benutzen können, Argumente aus und in den Stack mit Hilfe von Push- und Pop-Operationen zu transferieren.

Neben diesen Speicher-Adressiermöglichkeiten umfaßt die Architektur der Serie 32000 über 100 Grundbefehle. Der Befehlssatz ist symmetrisch, das heißt, jeder Befehl läßt sich mit jeder Adressierart sowie jeder Operandenlänge benutzen, z. B. Byte-, Wort- und Doppelwort-Format. Einige Instruktionen implementieren alle Arithmetikoperatoren der Sprache C (+, −, ∗, &, ×, %, !, ∧, ++, − und ~). Boolesche Zuweisungsinstruktionen werden für die relationalen Operatoren der Sprache C unterstützt (==, !=, <, <=, > und >=). Die Serie 32000 bietet die Extract- und Insert-Bitfeld-Instruktionen, um den Bitfeld-Datentyp der Sprache C unterstützen zu können, außerdem umfangreiche String- und Block-Move-Befehle zur Unterstützung der standardmäßigen C-Bibliotheksfunktionen. Daraus ergibt sich, daß Funktionen wie z. B. „Strlen" in lediglich vier Maschineninstruktionen implementiert sind.

Die Vorteile des symmetrischen Befehlssatzes der Serie 32000 beziehen sich nicht nur auf die Leistung des Betriebssystems, sondern auch auf Anwendungsprogramme, die in Hochsprachen geschrieben sind, z. B. C, Pascal und Fortran. Weil die Befehle und Adressierarten alle Anforderungen von Hochsprachen-Compilern erfüllen, können solche Compiler den Code sehr schnell generieren, außerdem ist er ziemlich kompakt und leicht fehlerfrei zu machen.

Für Implementierer und Endbenutzer von Unix-Systemen liegen folgende Vorteile auf der Hand: Weil Unix vornehmlich in C geschrieben ist, kann man mit der Serie 32000 eine höhere Gesamtsystemleistung erreichen als mit ähnlichen Mikroprozessoren, die diese Hochsprachenmerkmale nicht unterstützen.

Die richtige Implementierung des virtuellen Speichers

Bei der Auswahl eines Mikroprozessors für ein Unix-System sollten die Benutzer genau prüfen, wie die Architektur virtuelle Speicher unterstützt. Ein virtueller Speicher bedeutet, daß der Programmierer nicht mehr von den Eigenschaften des physikalischen Hauptspeichers eingeschränkt wird, weil die Umsetzung von logischen in physikalische Adressen einen uniformen logischen Adreßbereich für den Programmierer erreicht. Daraus ergibt sich, daß der gesamte Speicheradreßbereich des Prozessors dem Programm zur Verfügung

steht, obwohl lediglich ein Teil des physikalischen Adreßbereiches tatsächlich benutzt wird.

Der Demand-Paged-Zugriff auf den virtuellen Speicher, eine Methode, die üblicherweise bei Großrechnern oder Superminis, z. B. IBM 370 oder VAX von DEC, zu finden ist, teilt den physikalischen Speicher in „Seiten" (Pages) von gleicher Größe. Seiten, die das gerade auszuführende Programm benötigt, werden von der Platte auf den Hauptspeicher umgeladen. Bei der 32000-Familie überwacht der MMU-Baustein (Memory Management Unit), welche Seiten sich im Speicher befinden und welche auf der Platte sind. Jeder Speicherzugriff auf eine Page, die nicht im Speicher vorhanden ist, verursacht einen Prozessor-Interrupt, worauf die erforderlichen Seiten von der Platte in den Speicher umgeladen werden. Anschließend wird der Prozessor neu gestartet und der Befehl, der den Interrupt verursachte, abgearbeitet.

Der Vorteil des Demand-Paged-Zugriffes auf virtuelle Speicher ist, daß Anwendungsprogramme für große Maschinen mit 4, 6 oder mehr MByte Hauptspeicher auch auf kleineren Systemen mit lediglich einem MByte Speicher laufen können, ohne daß eine Änderung erforderlich wird. Dieser Vorteil ist insbesondere für Unix-Anwender wichtig, weil viele spezielle und technische Anwendungsprogramme, die unter Unix entwickelt wurden, auch für Supermini- oder Mainframe-Rechner mit virtuellem Speicher entstanden. Um diese Software auf kleinere, kostengünstigere Super-Mikrocomputer zu portieren, müssen diese auch virtuelle Speicherzugriffe unterstützen. Mit Ausnahme der 32000-Familie findet man bei den weitverbreiteten Mikroprozessortypen keine Berücksichtigung des virtuellen Speichers in der Architektur. Bei Systemen auf der Basis der Prozessoren 68000 oder 8086 benutzt man sehr unterschiedliche Implementierungen zur Speicherverwaltung. So verwendet man beispielsweise beim 80286 eine segmentierte Speicherverwaltung, bei der Blöcke unterschiedlicher Größe (maximal 64 KByte) im Speicherbereich abgebildet werden. Der Speicherverwaltungs-Baustein 68451, der zusammen mit der CPU 68010 verwendet wird, benutzt ebenfalls Speichersegmente variabler Größe.

Bild 3. Das Betriebssystem muß bei Segmentierung des Speichers die einzelnen Abschnitte immer wieder zuordnen, damit genügend Platz für ausgelagerte Segmente bleibt

Der segmentierte virtuelle Speicher neigt dazu, unwirtschaftlich zu sein, weil ein Hauptspeicher sehr schnell fragmentiert wird, wenn Speicherblöcke unterschiedlicher Größe umgeladen werden. Wenn die entstehenden Lücken im Hauptspeicher nicht groß genug sind, um das einzuladende Segment aufzunehmen, muß das Betriebssystem entweder zusätzliche Segmente auslagern, damit genug Platz zur Verfügung steht, oder existierende Segmente neu und kompakt ordnen (*Bild 3*). Darüber hinaus müssen immer komplette Segmente umgeladen werden. Daher sind große Hauptspeicher erforderlich, um auch große Programmsegmente benutzen zu können.

Der virtuelle Speicher mit Demand-Paged-Zugriff, wie er von dem MMU-Slaveprozessor NS32082 implementiert ist, macht Fragmentierung überflüssig, weil der Speicherbereich in gleich große Pages von jeweils 512 Byte eingeteilt ist. Jedes Programm bis zu einer Grenze von 16 MByte im logischen Adreßbereich der Serie 32000, das auf einem Mini- oder Mainframe-Computer mit Demand-Paged-Zugriff auf dem virtuellen Speicher ausführbar ist, kann auch auf einem System mit dem CPU-Baustein NS32032 laufen, ohne daß man sich um die Kapazität des zur Verfügung stehenden Speichers kümmern muß.

Wohl am wichtigsten für Unix-Anwender ist die Tatsache, daß die Implementierung des virtuellen Speichers bei der Serie 32000 fast identisch zu derjenigen beim Minicomputer VAX-11 ist. Der virtuelle Speicher mit Demand-Paged-Zugriff ist für das Programm und den Programmierer unsichtbar, damit braucht man sich über Kapazitätsprobleme oder die richtige Planung der Segmentierungsstrategie keine Gedanken zu machen. Die Speicherverwaltung wird von der Hardware sowie dem Betriebssystem übernommen. Der hohe Grad an Kompatibilität bei der Serie 32000 erlaubt daher leichtes Portieren von Anwendungssoftware.

Um mit Hilfe der CPU NS32032 eine Maschine mit virtuellem Speicher aufbauen zu können, muß man diese mit der Speicherverwaltungseinheit NS32082 ergänzen. Die beiden Chips können als eine einzige CPU aufgefaßt werden, die physikalische Speicheradressen erzeugt. Aufgrund der Beschränkungen, die sich aus der Chipgröße ergeben, wird die MMU separat gefertigt und arbeitet als Slave-Prozessor. Nach Aufnehmen und Decodieren einer Instruktion sendet die CPU die virtuellen Adressen der Operanden zur MMU. Die MMU untersucht jede Adresse, um festzustellen, ob sie im Hauptspeicher vorhanden ist. Falls dies der Fall ist, sendet die MMU die physikalische Adresse zum Speicher.

Wenn sich der Operand nicht im Speicher befindet, sendet die MMU ein Signal zum Unterbrechen der CPU-Aktivitäten. Die CPU stoppt die Ausführung der Instruktion und stellt mit Hilfe spezieller Hardware den Zustand der Register, z. B. Programmzähler oder Stack-Register, her, der vor Ausführung des Befehls bestand. Die virtuelle Speicherroutine des Betriebssystems wird anschließend aufgerufen, um das Laden einer Speicherseite zu initiieren. Diese Routine sorgt dafür, daß die benötigten Daten auf einem Peripheriespeicher gefunden und anschließend in den Hauptspeicher gebracht werden. Unbenutzte Daten aus dem Speicher werden zum Peripheriespeicher zurückgebracht, um nötigenfalls Platz zu schaffen. Während des Ladens der Seite kann die CPU andere Aufgaben ausführen. Nachdem die Seite geladen ist, wird die Instruktion, die den Verarbeitungsabbruch verursacht hat, neu gestartet, wofür das Betriebssystem alle nötigen Operationen ausführt.

Neben der Umsetzung von virtuellen Adressen in physikalische Adressen verfügt die MMU über weitere wichtige Eigenschaften. Eine davon ist der doppelte Adreßbereich. Zu jeder beliebigen Zeit lassen sich entweder ein oder zwei Adreßbereiche von der MMU unterstützen. In der Betriebsart mit einem Adreßbereich teilen sich die Benutzer diesen Speicher, indem das Betriebssystem gemeinsame Umsetztabellen benutzt. Im Doppelbereichsbetrieb hat jeder Benutzer und das Betriebssystem separate virtuelle Speicherbereiche von 16 MByte. Zwei Page-Table-Base-Pointer (PTB0 und PTB1) in der MMU zeigen auf den Startpunkt der Umsetztabellen, die ihrerseits dazu dienen, separate Benutzer- und Supervisor-Bereiche zu implementieren.

Die MMU verfügt über eine zweite wichtige Einrichtung, denn sie hält eine Referenzeintragung bei den Seitentabellen auf Ebene 1 und 2 sowie ein Modifizierungsbit für die Ebene 2. Die Referenz- und Modifizierungsbits werden von der MMU immer dann gesetzt, wenn auf eine Seite zugegriffen wurde oder eine Modifikation vorgenommen wurde. Das Referenzbit wird periodisch vom Betriebssystem abgefragt und zurückgesetzt, um eine Statistik über die Häufigkeit von Zugriffen auf die derzeit im Speicher befindlichen Seiten zu führen, so daß die am wenigsten benutzten Seiten ausgelagert werden können, wenn neue Seiten zu laden sind.

Weil Architektur und Implementierung aller Produkte der 32000-Familie sauber strukturiert sind, gibt es keine Probleme mit komplexen Betriebssystemen wie z. B. Unix. Die CPU NS32032 sowie die MMU NS32082 unterstützen Memory-Mapped-Dateien, das heißt, daß eine Datei an einen Abschnitt im Adreßbereich des Benutzerprozesses gebunden ist. Schreib- und Lesevorgänge im Speicher innerhalb dieses Bereiches betreffen den Inhalt des bestimmten Teiles einer Datei. Wenn Seitenfehler innerhalb dieses Bereiches auftreten, lassen sich bei Bedarf Seiten aus der deklarierten Plattendatei transferieren. Wenn die Datei lediglich für Lesen geöffnet ist, kann ein entsprechender Schutzmechanismus aktiviert werden. Jeder Leseversuch wird wie eine Segmentierungsverletzung behandelt. Wenn die Datei für Lesen und Schreiben geöffnet ist, können die Seiten modifiziert werden; die Plattendatei wird aktualisiert, wenn die physikalische Seite erneut verwendet wird, oder der Prozeß abschließt bzw. aus anderen Gründen eine vollständige Aktualisierung notwendig wird.

Adreßumsetzung im Detail

Für die virtuelle Speicherverwaltung müssen sowohl der virtuelle als auch der physikalische Adreßbereich in 32768 Seiten zu jeweils 512 Byte eingeteilt werden. Die Umsetzung von virtueller in physikalische Adresse erfolgt in den zwei Ebenen der Seitentabellen, die sich im Hauptspeicher befinden. Die Seitentabelle der Ebene 1 verfügt über 256 Eintragungen von jeweils 32 Bit Breite. Damit ergibt sich ein Gesamtumfang auf Ebene 1 von 1024 Byte.

Die oberen 8 Bit (Bit 16...32) der virtuellen Adresse werden zum indizierten Zugriff auf die Seitentabelle der Ebene 1 benutzt. Der Inhalt der Seitentabelle zeigt auf den Beginn von einer der 256 Seitentabellen der Ebene 2. Jede Tabelle der Ebene 2 enthält 128 Eintragungen mit jeweils 32 Bit. Die Bits 9...15 der virtuellen Adresse werden benutzt, um einen indizierten Zugriff auf die entsprechende Seitentabelle auf Ebene 2 durchzuführen. Der Inhalt der Seitentabelle auf Ebene 2 zeigt auf die tatsächliche physikalische Seite.

Auf den ersten Blick scheint es, daß dieses Schema einen großen Teil des zur Verfügung stehenden Speicherbereiches dazu benötigt, die Tabellen abzuspeichern. Der Overhead ist allerdings außerordentlich klein: Ein Maximalsystem mit 16 MByte virtuellem Speicher benutzt eine Seitentabelle von 1024 Byte + 256 Seitentabellen mit jeweils 152 Byte, um insgesamt 132 096 Byte Eintragungen abzuspeichern. Davon können die Seitentabellen der Ebene 1 (131 072 Byte) in oder aus dem Hauptspeicher transferiert werden. Die Anzahl der Seitentabelleneintragungen läßt sich reduzieren, wenn das System nicht die vollen 16 MByte Adreßbereich benötigt.

Der Umsetzprozeß startet, wenn die MMU eine virtuelle Adresse von der CPU erhält. Die MMU vergleicht die oberen 15 Bit der virtuellen Adresse mit den 32 Eintragungen in dem Umsetzpuffer. Wenn die Adresse sich im Puffer befindet, was zu 98 % der Fälle auftritt, steht die physikalische Adresse in lediglich einem Taktzyklus zur Verfügung.

Bild 4 zeigt den Umsetzprozeß, der auftritt, wenn die Adresse sich nicht im Umsetzpuffer befindet. Die MMU benutzt ihre Seitentabellen-Basisregister (PTB0 oder PTB1) in Supervisor- oder Benutzerbetrieb, die vom Betriebssystem gesteuert werden, um die Stabadresse

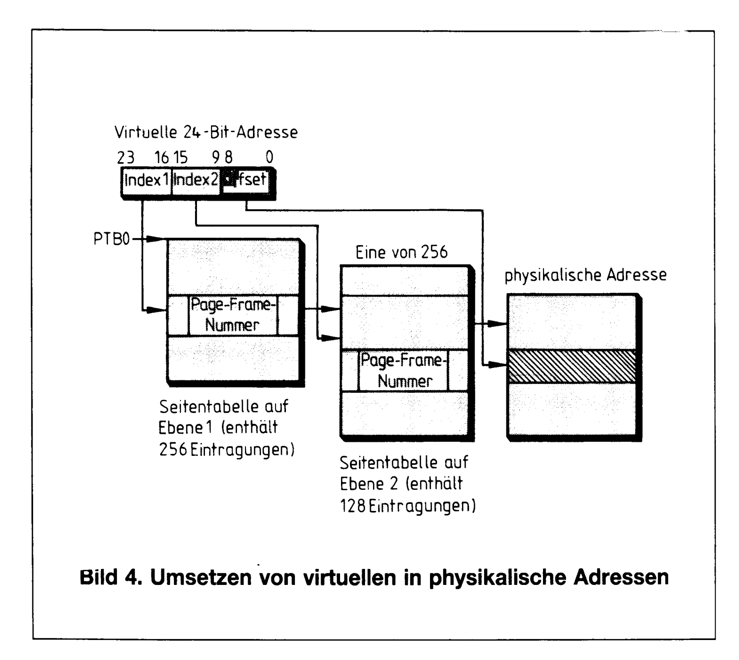


Bild 4. Umsetzen von virtuellen in physikalische Adressen

der Seitentabelle für die Ebene 1 im Hauptspeicher zu plazieren.

Die MMU nimmt die oberen 8 Bit der virtuellen Adresse, multipliziert die Zahl mit 4 und addiert daraufhin den Wert zur Startadresse der Seitentabelle für die Ebene 1. Dies ist der Indizierungsvorgang für die Seitentabelle der Ebene 1.

Jede 32-Bit-Seitentabellen-Eintragung der Ebene 1 enthält eine 16-Bit-Seiten-Framezahl, die von der MMU benutzt wird, um eine der 256 Seitentabellen der Ebene 2 zu lokalisieren. Die MMU nimmt die zweiten oberen 7 Bit der virtuellen Adresse, multipliziert diese Zahl mit 4 und addiert sie zur Seiten-Frame, die sich bei indiziertem Zugriff auf die Seitentabelle der Ebene 1 ergibt, um einen der 128 Eintragungen der Seitentabelle für die Ebene 2 zu lokalisieren. Die MMU nimmt anschließend die 15-Bit-Seiten-Framezahl aus den Eintragungen in die Tabelle der Ebene 2 und hängt die unveränderten unteren 9 Bits der virtuellen Adresse an, um die physikalische 24-Bit-Adresse zu bestimmen. Die MMU aktualisiert daraufhin den Umsetzpuffer mit der neuen physikalischen Adresse. Die vollständige Umsetzung erfolgt innerhalb von 16 Taktzyklen.

Unterstützung der Fließkommafunktionen

Das Betriebssystem Unix ist ein Produkt der akademischen, technischen und wissenschaftlichen Umgebung. Dies bedeutet, daß ein großer Teil der existierenden Unix-Anwendungsprogramme auf ausreichend leistungsfähige Fließkomma-Arithmetikprogramme angewiesen ist. Aus diesem Grund steht innerhalb der 32000-Familie die Fließkommaeinheit (FPU) NS32081 zur Verfügung, mit der sich IEEE-kompatible Fließkommaoperationen mit einer Breite von 32 Bit und 64 Bit ausführen lassen. Um der Entwicklungsphilosophie der Serie 32000 gerecht zu werden, arbeiten die Fließkommainstruktionen mit jeder beliebigen Adressierart.

Die CPU und FPU arbeiten so zusammen, daß dies für das Benutzerprogramm völlig transparent ist. Wenn die CPU einen Fließkommabefehl erkennt, wird ein spezielles Protokoll aufgerufen, das die Befehls-Opcodes und Operanden zur FPU transferiert. Die FPU führt die erforderlichen Berechnungen aus und informiert die CPU bei deren Abschluß. Daraufhin gibt die CPU der FPU ein Strobesignal, um Status- und Resultats-Informationen zu erhalten. Das Statuswort aktualisiert das Statusregister der CPU. Eine Trap-Behandlungsroutine läßt sich bei nichtnormalen Situationen aufrufen, z. B. wenn ein Überlauf auftritt.

Alle Daten- und Befehlstransfers zwischen CPU und FPU erfolgen über einen 16-Bit-Bus, der die beiden Bauelemente miteinander verbindet. Die Transfers werden in zwei Taktzyklen ausgeführt.

Die CPU enthält ein spezielles Konfigurationsregister, das das Vorliegen oder Fehlen des FPU-Bausteines bemerkt. Wenn die CPU nicht für die FPU konfiguriert ist und gleichzeitig ein Fließkommabefehl vorliegt, dann reagiert die CPU damit, daß ein Trap für eine undefinierte Instruktion ausgeführt wird. Dieser Trap erlaubt die Software-Emulation von Fließkommabefehlen bei Anwendungen, in denen die Verarbeitungszeit nicht kritisch ist.

Tabelle 1 zeigt die Unterschiede in der Ausführungszeit zwischen Fließkommabefehlen, die von der FPU, und die gleichen Instruktionen, die vom Software-Fließkommapaket (FPP) ausgeführt werden. In den meisten Fällen liegt die FPU-Ausführungszeit bei lediglich 1/100 der Software-Routine. Alle Zeiten gelten für den gesamten Befehlszyklus einschließlich CPU-Operandenadressierung und CPU-/FPU-Kommunikation über das Slave-Protokoll.

Tabelle der Fließkomma-Operations-Ausführungszeiten (10 MHz) in µs

mit FPU	Addition/ Subtraktion		Multiplikation		Division	
	Single	Double	Single	Double	Single	Double
Register/Register	7,4	7,4	4,8	6,2	8,9	11,9
Speicher/Register	8,5	9,6	5,9	8,4	10,01	4,1
Register/Speicher	9,5	11,8	7,0	10,6	11,1	16,3
Speicher/Speicher	11,1	14,4	8,5	13,2	12,6	18,9
mit FPS	1330	1500	1300	1700	1300	1950

Fließkomma-Operationen im Detail

Wie aus *Bild* 5 hervorgeht, besteht die FPU aus drei architektonischen Blöcken: Steuereinheit, die für das Steuern des Ausführungsflusses in der Arithmetikfunktion zuständig ist, Ausführungseinheit, die spezielle Parallelhardware benutzt, um schnelle Ausführung der Arithmetikoperationen zu gewährleisten, und schließlich die Interface- sowie Speichereinheit, die für die Verwaltung des CPU-FPU-Kommunikationsprotokolls verantwortlich ist.

Die Steuereinheit kümmert sich um die Datentransfers zwischen den Ausführungs- und den Interface-Einheiten und um die Aktivierung aller Unter-Funktionsblöcke der Ausführungseinheit. Ein wichtiges Ziel bei der Entwicklung war, daß sich die Unter-Funktionsblöcke parallel aktivieren lassen, was sich durch die Verwendung eines horizontalen Mikrocodes mit separaten Feldern für jede Einheit erreichen ließ.

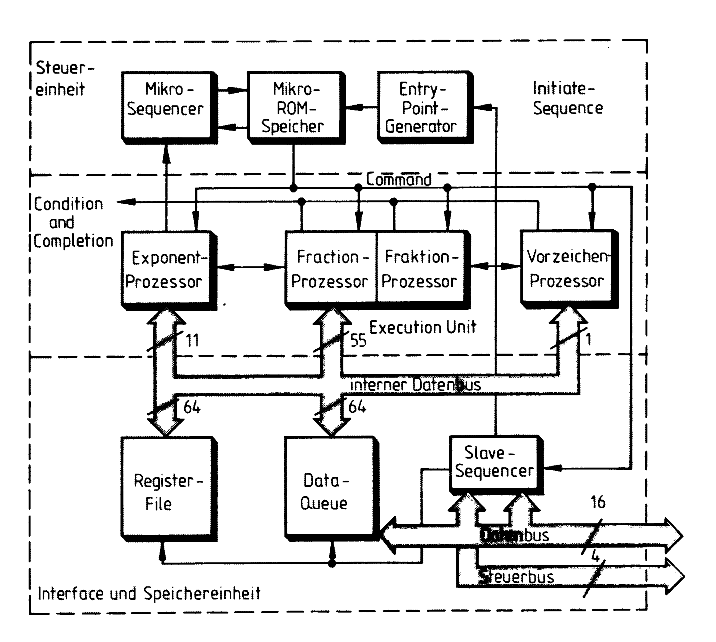


Bild 5. Die FPU besteht aus drei Funktionsblöcken

Ein Opcode-Decodierer steuert den gesamten Kommagenerator, der den ersten Block in der Steuereinheit darstellt. Dieser wird auch „Eingangs-Punkt-Generator" genannt und definiert den Beginn einer „Mikrocode-Sequenz. Dieser Block ist an ein ROM angeschlossen, welches 450 Steuerwörter mit jeweils 20 Bit enthält. Jedes Wort besteht aus fünf Feldern, davon sind drei für die Steuerung der Ausführung in den Hauptfunktionsblöcken der Einheit, eines für die Synchronisation und eines für die sequentielle Abarbeitung der nächsten Operation. Der Sequenzerblock enthält einen Teil der Adresse für die nächste Mikroinstruktion. Diese Methode vermeidet lange, zeitaufwendige „Prüf- und Verzweigungs"-Sequenzen im Mikrocode und sorgt zum größten Teil für die außerordentlich hohen Durchsatzwerte des FPU-Bausteins.

Der FPU-Baustein NS32081 ist auch aufgrund des Konzeptes der Ausführungseinheit sehr schnell. Diese Einheit besteht aus drei Recheneinheiten (für Vorzeichen, Exponenten und Fraktion) sowie einer Steuereinheit. Diese vier Einheiten arbeiten parallel, wodurch sich die kurzen Ausführungszeiten ergeben. Außerdem gibt es einige wichtige Algorithmen, die in Hardware implementiert sind, wodurch sich auch rechenintensive Betriebssysteme, z. B. Unix, problemlos unterstützen lassen.

Virtueller Speicher mit Demand-Paged-Zugriff für Genix

Für einen Implementierer von Unix stellt dieses Betriebssystem gleichzeitig auch den besten Test für die Leistung eines Mikroprozessors dar. Genix ist National Semiconductors Portierung von 4.1 Bsd-Unix, die erste Unix-Version für einen Mikroprozessor, der virtuelle Speicher mit Demand-Paged-Zugriff unterstützt.

Seitentabellen auf zwei Ebenen

National Semiconductor schrieb den Code für den virtuellen Speicher von 4.1 Bsd völlig neu, um die Architektur der MMU NS32081 optimal ausnutzen zu können. Ein Seitentabellen-Basisregister wird für Kernbetriebsart benutzt und enthält die Adresse im physikalischen Speicher der Seitentabelle in der Ebene 1. Diese Tabelle wiederum enthält 256 Seitentabellen-Eintragungen. Jede Seite steuert den Zugriff auf einen Bereich von 128 Seiten im Benutzeradreßbereich, in dem sie die Abbildung einer Seitentabelle für diesen Bereich definiert. Die Seitentabelle auf Ebene 2, die 128 Eintragungen besitzt, definiert die Abbildung für die 128 individuellen Seiten im entsprechenden Adreßbereich. Die Seitentabellen auf Ebene 2 müssen nicht unbedingt vollständig sein. Lediglich, wenn Seiten in diesem Adreßbereich existieren, muß eine Tabelle auf Ebene 2 kreiert werden, im anderen Fall ist die Seitentabellen-Eintragung durch eine 0 gekennzeichnet.

Das zweite Seitentabellen-Basisregister der MMU wird zum Abbilden des Adreßbereiches in der Benutzerbetriebsart in ähnlicher Weise wie dies vorher beschrieben ist benutzt. Das Seitentabellen-Basisregister des Benutzers zeigt auf die eigene Seitentabelle in Ebene 1, die wiederum die Tabellen und Adreßbereichsseiten der Ebene 2 abbildet. Die MMU bezieht sich automatisch auf die Kern- oder Benutzer-Map, je nach dem laufenden Prozessorstatus und dem Befehl.

Format einer Seitentabellen-Eintragung

Bild 6 zeigt das Format einer Seitentabellen-Eintragung. Jede Eintragung auf der Seitentabelle der Ebene 2

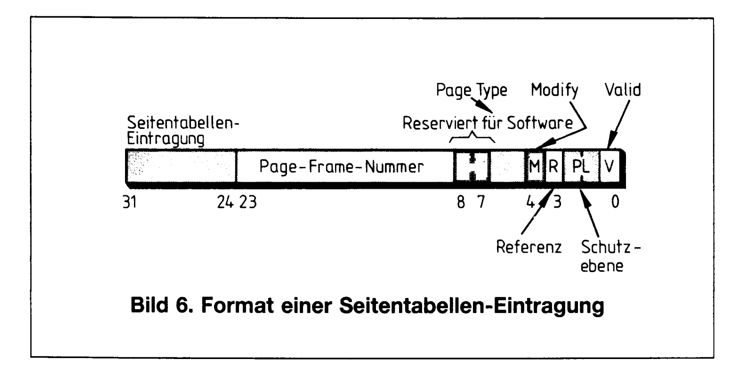


Bild 6. Format einer Seitentabellen-Eintragung

enthält ein Gültigkeitsbit, das, wenn es gesetzt ist, anzeigt, ob die virtuelle Seite im physikalischen Speicher existiert. Außerdem enthält sie die physikalische Seitenzahl, die zu dessen virtueller Seite korrespondiert, zwei software-definierbare Bits, zwei Schutzebenenbits, ein Referenzbit sowie ein Modifizierungsbit werden automatisch von der MMU verwaltet und können außerdem über den Kern gesetzt oder zurückgesetzt werden. Wenn das Gültigkeitsbit nicht gesetzt wurde, ist das Format für den Rest der Eintragung vollständig für die Benutzung durch Software reserviert. Das Format der Seitentabellen-Eintragung auf Ebene 1 entspricht dem der Ebene 2 mit der Ausnahme, daß das Modifizierungsbit nicht bei Eintragungen in die Tabelle der Ebene 1 berücksichtigt wird.

Die beiden Schutzebenenbits werden zur Steuerung von Zugriffen auf die Seiten, die in den Seitentabellen enthalten sind, benutzt. Die Beziehung zwischen Benutzer, Kern und Schutzebenenbits ist folgende:

Benutzer:
00 kein Zugriff
01 kein Zugriff
10 nur Lesen
11 Lesen/Schreiben
Kern:
00 nur Lesen
01 Schreiben/Lesen
10 Schreiben/Lesen
11 Schreiben/Lesen

Benutzerprozeß-Page-Mapping

Page-Mapping unter dem Betriebssystem Genix unterstützt Text-, Daten- und Stacksegmente, wie sie in *Bild 7* zu erkennen sind. Üblicherweise bestehen die ersten Seiten eines Benutzerprozesses aus ausführbarem Textcode. Diese sind geschützt, so daß sie nur lesbar sind. Der Text reicht immer bis zur nächsten Seitengrenze, wobei die letzte Seite des Textes gegen Schreibvorgänge geschützt sein muß. Wenn der Benutzer versucht, eine Instruktion auszuführen, die in diese Seite schreiben will, wird das Schutzfehler-Statusbit im MMU-Statusregister gesetzt. Dies verursacht einen Interrupt, der vom Kern verarbeitet wird.

Die Daten folgen immer dem nur bei Lesebetrieb ausführbaren Code und beginnen mit der nächsten Seitengrenze; die Datenseiten sind gegen Lesen und Schreiben geschützt. Nach den Daten kommt typischerweise ein größerer gegen Lesen und Schreiben geschützter Adreßbereich, an dessen Spitze der Benutzerstack angeordnet ist. Wenn der Stack während eines „exec"-Aufrufes konfiguriert wird, reserviert der Kern eine nicht abgebildete Seite oberhalb des Stacks, um zu ermöglichen, daß der Überlauf des Stacks erkennbar ist. Außerdem werden genügend Seiten für den Stack abgebildet, um die exec-Argumente und die Umgebungsinformationen abzulegen. Der Page-Fault-Handler erweitert, wenn erforderlich, während der Programmausführung den Stack.

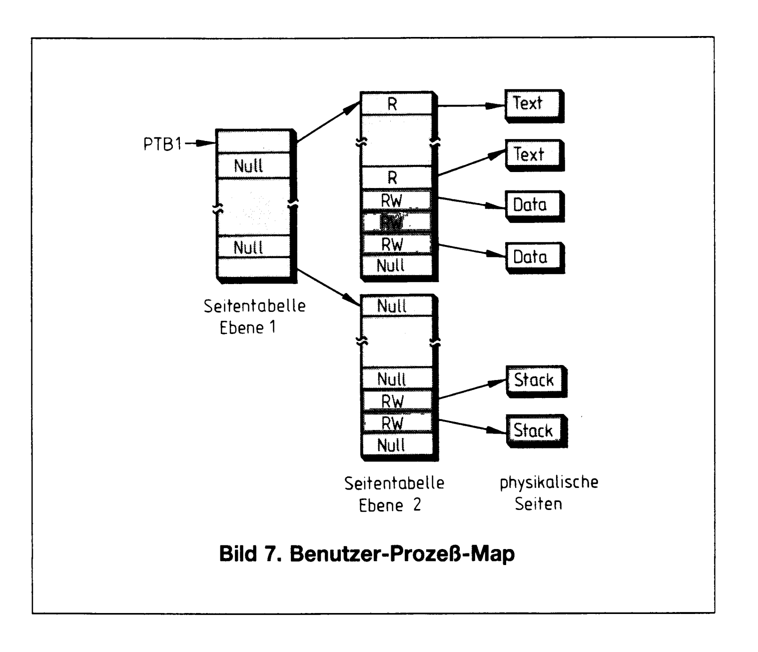


Bild 7. Benutzer-Prozeß-Map

Wenn ein Benutzerprogramm zum ersten Mal ausgeführt wird, erzeugt der Kern eine File-Map (*Bild 8*), indem die Seitentabellen-Eintragungen zugeordnet und das Programm in den Speicher eingelesen werden. Wenn die File-Map aufgestellt ist, wird die Map des Prozesses initialisiert. Für die Seiten, die nur gelesen werden dürfen, zeigen die Seitentabellen-Eintragungen des Prozesses auf die gleichen physikalischen Seiten, in die auch die Datei gelesen wurde. Die Eintragungen der Seitentabelle auf Ebene 2 für beschreibbare Seiten des Prozesses tragen die Bezeichnung „SPT" und enthalten einen Index für die File-Map sowie eine Seite in dieser Map, die die ursprünglichen Daten enthält. Wenn der Prozeß erstmals auf Daten mit der Markierung „SPT" zugreift, wird eine Page-Trap generiert. Der Kern reagiert darauf durch Kopieren der Originaldaten auf eine neue Seite für exklusive Benutzung durch den Prozeß, markiert den Eintrag in die Seitentabelle auf Ebene 2 mit „MEM" und startet das Benutzerprogramm. Im Betriebssystem Genix können alle ausführbaren Objektdateien auf diese Weise verteilt werden.

Jedes Mal, wenn ein Prozeß eine ausführbare Datei verteilt, wird ein Zähler für die Datei-Map inkrementiert. Die Datei bleibt abgebildet und steht für die Aufteilung bereit, bis der Zähler auf Null zurückgezählt ist und die Seite von einem anderen Prozeß benötigt wird. Die File-Map, der Text und die Datenseiten bleiben im Speicher, auch wenn das Programm selten von zwei Prozessen gleichzeitig ausgeführt wird.

Kern-Page-Mapping

Der Kernadreßbereich umfaßt den gesamten Kerncode, den Rest des physikalischen Speichers, die Device-Register und Platz für Datenstrukturen, die während der Laufzeit anfallen. Die erste dieser Datenstrukturen ist der Bereich für die Seitentabellen des Kerns auf der Ebene 1, darauf folgen die Tabellen der Ebene 1 über die Benutzerprozesse. Diese werden gefolgt von verschiedenen Seitentabellen für die Abbildung von ausführbaren Dateien auf der Ebene 1. Die Seitentabellen auf der Ebene 1 werden im Speicher immer bereitgehalten. Der gesamte Kernadreßbereich ist durch den Benutzer-Mode-Prozeß gegen Zugriff geschützt.

Seitentypen

Die beiden software-definierbaren Bits in der Seitentabellen-Eintragung erlauben es dem Kern, bis zu acht verschiedene Typen von Seitentabellen-Eintragungen zu definieren. Genix definiert drei Arten von Valid-Pages (virtuelle Seiten, die im physikalischen Speicher existieren) und vier Arten von Invalid-Pages (virtuelle Seiten, die nicht im physikalischen Speicher enthalten sind). Ein Page-Typ ist undefiniert.

Zu den Valid-Pages gehören folgende Typen:

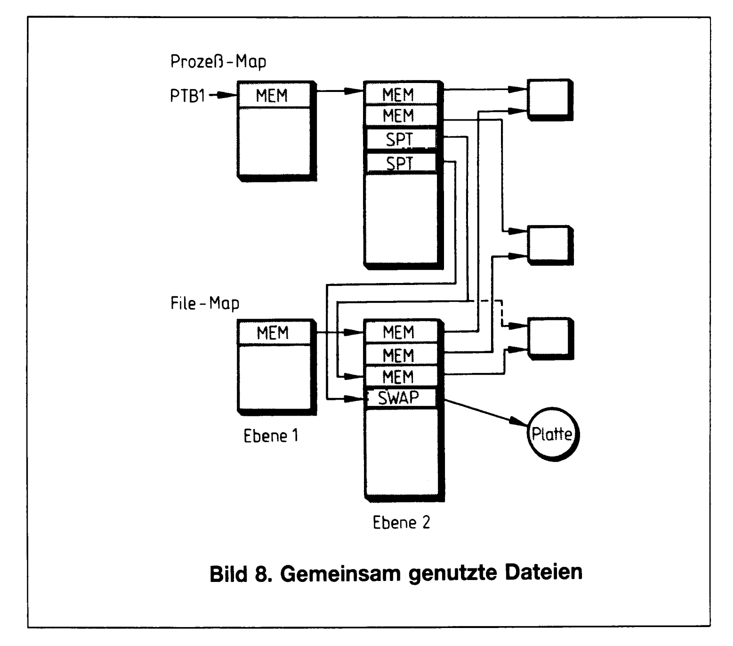


Bild 8. Gemeinsam genutzte Dateien

- MEM: bildet eine Seite des virtuellen Adreßbereiches im physikalischen Speicher ab. Der Kern bestimmt die physikalische Seite.
- LOCK: ähnlich wie MEM, zeigt allerdings dem Kern, daß diese Page des virtuellen Adreßbereiches nicht ausgelagert werden soll. Das Betriebssystem Genix unterstützt einen Systemaufruf, der es dem Benutzer erlaubt, den LOCK-Seitentyp zu spezifizieren.
- SPY: bildet eine virtuelle Seite einer spezifizierten physikalischen Adresse ab. Dieser Typ wird vom Kern dazu benutzt, Device-Register, die bekannte physikalische Adressen einnehmen, im virtuellen Adreßbereich des Kern abzubilden. Das Betriebssystem Genix stellt dafür einen Systemaufruf zur Verfügung, der den SPY-Seitentyp anspricht.

Die Invalid-Page-Typen sind:

- NULL: die Seite ist nicht abgebildet.
- SWAP: die Seite ist auf die Platte geschrieben.
- IOP: es ist ein E/A-Vorgang in Bearbeitung, die zu verarbeitende Seite wird gerade hinein- oder heraustransferiert.
- SPT: Diese Eintragung enthält einen Pointer auf Page-Tabellen-Eintragung einer anderen Page-Tabelle, die die Mapping-Information für die virtuelle Seite enthält. Der Typ SPT wird vom Kern zur Implementierung von gemeinsam genutztem Text benutzt.

Page-Fault-Handling

Wenn entweder der Kern oder ein Benutzerprozeß versucht, Zugriff auf virtuelle Adressen zu erlangen, die nicht im physikalischen Speicher enthalten sind, erzeugt die MMU ein Page-Fault-Trap, das den Page-Fault-Handler des Kerns aufruft. Der Page-Fault-Handler besitzt die Information aus dem MMU-Statusregister und dem Fehler-/Ungültigkeits-Adreßregister, das die virtuelle Adresse enthält, die die Seitenfehler erzeugt und anzeigt, ob die Adresse im Benutzer- oder im Kern-Adreßbereich enthalten ist. Neben dem Prüfen der Legalität eines Zugriffsversuches wird ein Seitenfehler aus dem Kern-Adreßbereich in gleicher Weise verarbeitet, wie ein Fehler aus dem Benutzer-Adreßbereich.

Der größte Teil des Page-Fault-Handling entsteht durch einem legalen Zugriff auf eine virtuelle Adresse, die nicht im physikalischen Speicher existiert. Der Kern weist dem Prozeß eine Seite im physikalischen Speicher zu und liest, falls möglich, die Daten in diese Seite. Daraufhin aktualisiert der Kern die Page-Map des Prozesses und kehrt von der Trap zurück. Bei der Rückkehr von der Trap startet die CPU den Befehl, der die Trap auslöste, und die Ausführung des Prozesses wird fortgesetzt.

Wie bereits vorher beschrieben, definiert der Kern vier ungültige Typen der Seitentabellen-Eintragung. Dafür ist die jeweilige Fehlerbehandlung unterschiedlich. Wenn die Seitentabellen-Eintragung mit „SWAP“ gekennzeichnet ist, gibt die Eintragung einen Platz im Platten-Paging-Bereich der Prozeßseite an. Um den Feh-

ler zu beseitigen, liest der Kern die Platte in die Speicherseite, nimmt, falls erforderlich, ein Rescheduling des E/A-Bereiches vor, startet daraufhin erneut das Benutzerprogramm.

Wenn die Seitentabellen-Eintragung mit „Null" gekennzeichnet ist, bedeutet dies, daß auf die virtuelle Seite erstmals ein Zugriff versucht wird. Der Kern weist eine Seite im Speicher zu, die auf Null initialisiert wurde, aktualisiert die Prozeß-Page-Map und kehrt zum Programm zurück. Dieser Vorgang erfolgt, wenn der Stack erweitert wird oder erstmals Zugriff auf eine Seite im uninitialisierten Datenbereich des Benutzers geschieht. In allen anderen Fällen erzeugt der Kern ein Bus-Fehlersignal.

Wenn die Seitentabellen-Eintragung mit „SPT" gekennzeichnet ist, handelt es sich bei der virtuellen Seite um eine gemeinsame Textdatei. Die Seiteneintragung enthält einen Index, der die entsprechende File-Map und Seitentabellen-Eintragung anzeigt. Falls die Seitentabellen-Eintragung anzeigt, daß sich die Seite im Speicher befindet, wird die Seitentabellen-Eintragung des Benutzers so verändert, daß sie auf dieser Seite abgebildet ist, und daraufhin das Benutzerprogramm erneut gestartet. Falls die Seite nicht im Speicher ist, wird sie dorthin transferiert, anschließend die File-Map aktualisiert, die Benutzer-Seitentabellen-Eintragung auf den neuesten Stand gebracht und schließlich das Benutzerprogramm erneut gestartet.

Wenn die Kennzeichnung „IOP" vorliegt, werden die Daten für die virtuelle Seite gerade transferiert. Der Kern wartet, bis der E/A-Prozeß abgeschlossen ist, nimmt, falls erforderlich, ein Rescheduling vor und prüft anschließend die Seitentabellen-Eintragung. Falls sich die Seite jetzt im Speicher befindet, wird das Benutzerprogramm gestartet, falls nicht, wird die Seite in den Speicher gebracht.

Behandlung freier Seiten

Der Kern führt zwei verknüpfte Listen freier Seiten, eine für diejenigen, von denen bekannt ist, daß sie auf Null gebracht sind, die andere von sogenannten „dirty Pages". Freigewordene Seiten kommen zur Liste der „dirty Pages". Wenn Zeit vorhanden ist, entnimmt der Kern Seiten aus dieser Liste, setzt sie auf Null und übernimmt sie in die Liste der auf Null gesetzten Seiten. Speicherseiten werden aus diesen beiden Listen zugewiesen, je nachdem, ob eine Seite benötigt wird, in die Daten zu lesen sind, oder ob der Kern eine Seite für den Benutzerstack zuweist, die auf Null gesetzt sein muß. Falls die Liste der Null-Seiten leer ist, holt sich der Kern eine „dirty Page" und setzt sie während der Zuweisung auf Null.

Seiten-Ersatzalgorithmus

Wenn physikalischer Speicherbereich zur Lösung eines Seitenfehlers benötigt wird, aber keine Speicherkapazität zur Verfügung steht, muß durch Auslagerung von Seiten entsprechender Platz geschaffen werden. Der Seiten-Ersatzalgorithmus wählt eine Seite, die aus dem Speicher zu bringen ist, und aktualisiert die Seitentabelle des Prozesses bzw. der Prozesse, die diese Seite benutzen. Der Algorithmus arbeitet zusammen mit der „Core-Status-Tabelle" des Kerns, die für jede Seite des physikalischen Bereiches eine Eintragung besitzt. Das Programm benutzt Informationen, die in dieser Tabelle enthalten sind, um die Seitentabellen-Eintragung auf der Ebene 2 für jede physikalische Seite zu lokalisieren. Das Programm läuft durch die Seitentabellen-Eintragungen und prüft das Referenzbit jeder Eintragung. Wenn dies gesetzt ist (was anzeigt, daß erst kürzlich auf die Seite zugegriffen wurde), setzt das Programm dieses Bit zurück. Wenn keine Seite gefunden wurde, auf die kein Zugriff erfolgte, läuft dieser Algorithmus erneut ab. Falls in der Zwischenzeit auf eine Seite kein Zugriff erfolgte, wird diese Seite für die Auslagerung ausgewählt.

Bevor die Seite auf die Platte geschrieben wird, prüft der Kern das „Modified"-Bit der Seitentabellen-Eintragungen. Falls dies anzeigt, daß die Seite noch nicht verändert worden ist, seitdem sie zuletzt auf die Platte geschrieben wurde, kann der Kern die Seite unmittelbar aus dem Speicher entfernen. Falls eine Veränderung vorgenommen wurde, reserviert der Kern einen Platz auf der Platte, aktualisiert die Map des Prozesses, der die Seite benutzt, und sorgt dafür, daß spätere Zugriffe auf die Seite ungültig sind. Anschließend schreibt der Kern die Seite auf die Platte, so daß sie für andere Zwecke zur Verfügung steht.

Aufgrund der Definition kann der Algorithmus keine Seiten auswählen, die mit „LOCK" gekennzeichnet sind, außerdem vermeidet er die Auswahl von gemeinsamen Seiten. Falls dies einmal der Fall sein sollte, müssen die File-Map sowie die Maps aller Prozesse, die diese Seiten benutzen, aktualisiert werden.

Wenn die Anzahl der Paging-Vorgänge einen im Kern vorgegebenen Wert überschreitet oder wenn während einer spezifizierten Anzahl von Sekunden nur wenig freier Speicherbereich zur Verfügung steht, macht der Scheduler eine größere Anzahl von Seiten auf einmal frei, indem er den gesamten Prozeß auslagert. Die gemeinsamen Dateien, die immer noch im Speicher, aber nicht in Benutzung sind, werden immer vor den Benutzerprozessen ausgelagert. Außerdem wurden Vorkehrungen getroffen, daß ein Prozeß teilweise ausgelagert wird, falls die Speicherbedingungen dies verlangen. Dadurch erreicht man, daß ein größeres Programm nicht vollständig aus dem Speicher entfernt wird, nur um ein kleines Programm ablaufen zu lassen. Der Scheduler sucht die Seitentabelle des ausgewählten Prozesses ab und schreibt die „Unlocked"-Seiten dieses Prozesses auf die Platte. Der Scheduler benutzt für die Aktualisierung

der Prozeß-Map und die Zuweisung von Plätzen im Paging-Bereich das Verfahren des Seiten-Ersatzalgorithmus.

Optimale Systemleistung und Funktionalität

Die Strategie bei der Portierung von Unix ist die Optimierung des Systems, um alle Vorteile der Hardware zu nutzen, aber trotzdem so nah wie möglich an der Version Berkeley 4.1 zu bleiben. Dazu nutzt Genix zwei wichtige Hardware-Merkmale der 32000-Familie aus: die MMU und die FPU. Das Betriebssystem Genix ist kompatibel zu Berkeley 4.1, in dem ein vorgegebenes Merkmal der Hardware oder Software für die Benutzerprozesse völlig unsichtbar ist, oder in dem dieses Merkmal in einem Bereich isoliert wird, der durch derzeitige Berkeley-4.1-Software nicht berührt wird. Daraus ergibt sich, daß die Funktion von Genix mit derjenigen von Berkeley 4.1 gleich ist, während die neue Software, die das Portierungsteam geschrieben hat, alle Vorteile der 32000-Merkmale ausnutzt.

Der Genix-Debugger „ddt" ist dafür ein gutes Beispiel. Hiermit stehen dem Benutzer Breakpoint-Einzelschritt- und Stack-Backtrace-Funktionen auch auf der Interruptebene zur Verfügung. ddt kann gleichzeitig in Benutzer- und Supervisor-Betriebsart laufen, so daß sich Fehler im Interadreßbereich finden lassen. Zum Beispiel kann ein Programmierer einen Systemaufruf vom Benutzerprozeß in den Kern und wieder von dort hinaus verfolgen.

„ddt" arbeitet in zwei Betriebsarten, nämlich als normaler Benutzer-Prozeß-Debugger und als Remote-Debugger. Als Remote-Debugger läuft ddt auf einer Host-Maschine und kommuniziert über eine RS-232-Verbindung mit einem ROM-Monitor in der Zielmaschine.

„ddt" berücksichtigt auch die beiden Speicher-Referenz-Breakpoint-Register der MMU. Diese Register erlauben es einem Benutzer, bei Zugriff auf einen Teil des Benutzeradreßbereiches zu unterbrechen, was bedeutet, daß der Prozessor anhalten kann, wenn eine Variable sich verändert.

Die Portierung von Berkeley 4.1 auf die CPU NS32032 resultiert in drei neuen Systemaufrufen, die die Vorteile dieser Architektur ausnutzen: vspy, vlock und vmap.

Der Systemaufruf vspy bildet eine Seite aus einem Benutzer-Prozeß-Adreßbereich unter einer gegebenen physikalischen Adresse ab. Im Gegensatz zu Systemaufrufen der Version 7 „phys" wird die bestimmte physikalische Seite ausschließlich in einem Prozeß abgebildet. vspy ist sehr nützlich für die Abbildung von E/A-Devices, z. B. einem Bit-Mapped-Bildschirm in einem Adreßbereich des Prozesses. Das Format des Aufrufes ist:

vspy (virtual_address, physical_address, read_write_flag); der Systemaufruf vlock sperrt oder öffnet eine spezielle Seite eines privilegierten Prozesses im Speicher. Gesperrte Seiten können nicht ausgelagert werden, was für Echtzeit-Anwendungen sehr wichtig ist. Das Format des Aufrufes ist:

vlock (lockflag, virtual_address);

der Systemaufruf vmap ist ein privilegierter Aufruf, der zur Abbildung oder zum Kopieren einer Seite des Adreßbereiches eines anderen Prozesses in einen laufenden Adreßbereich dient. vmap wird durch „ps" sowie „w" benutzt, um Kommando-Argumente für andere Prozesse bereitzustellen. Das Format des Aufrufes ist:

vmap (function, this_process_address, other_process_address, pid).

Die MMU unterstützt darüber hinaus auch das Programmfluß-Tracing: Die Speicherverwaltungseinheit kann bei nichtsequentieller Befehlsausführung unterbrechen, das bedeutet, daß ddt Programmsteuerdaten übernehmen kann. Die MMU kann auch die letzten beiden nichtsequentiellen Befehlsverzweigungen aufzeichnen sowie die letzten beiden Stände des Programmzählers, was sehr nützlich ist, wenn man die Vergangenheit der Programmabarbeitung verfolgen muß. Auf diese Weise kann ein Benutzer einen kleinen Ausschnitt aus der Programmaktivität vor einer Ausnahmebedingung, z. B. dem Verstoß gegen die Segmentierung, verfolgen. Die Flow-Register werden nicht zurückgesetzt oder gelöscht, so daß im Remote-Betrieb ddt die letzten beiden Adressen noch angezeigen kann, auch wenn das System bereits in einem fehlerhaften Zustand ist.

Das Betriebssystem Genix unterstützt die standardmäßigen Dienstprogramme von Berkeley 4.1: die C-Shell, den Bildschirm-Editor vi, die Textvorbereitungs-Einrichtung uucp, nroff und troff, außerdem gehören dazu ein 32000-Assembler, -Lader, Debugger für die symbolische Adressierung, C-Compiler, run-Time-Libraries sowie ein optioneller Pascal-Compiler. Das Betriebssystem Genix umfaßt Treiber für CRT, Paralleldrucker, Platte und Kassetten-Bandlaufwerk, außerdem bietet es vollständige Kompatibilität mit den Slave-Prozessoren der Serie 32000 für die Speicherverwaltung und Fließkomma-Arithmetik.

National Semiconductor wählte Berkeley-Unix, weil dieses Betriebssystem große Verbreitung gefunden hat und die seitenorientierte Zugriffssteuerung (Demand-Paging) für den Speicher unterstützt. So ist beispielsweise die Portierung eines VAX-Anwendungsprogrammes auf die Genix-Umgebung völlig problemlos.

Ing. (grad.) Günther Hausmann, geboren in Göttingen, studierte Kybernetik an der FH Furtwangen. 1976 trat er in die Digital Equipment GmbH in München ein, wo er als Central European Training Marketing Manager tätig war. Seit 1984 ist er als European Software Product Marketing Manager bei National Semiconductor GmbH beschäftigt. Er ist verheiratet, hat zwei Kinder, und in seiner Freizeit geht er am liebsten mit seinem Dyas Rennboot segeln.



Jean-Claude Mathon

Am oberen Ende der Leistungsskala:

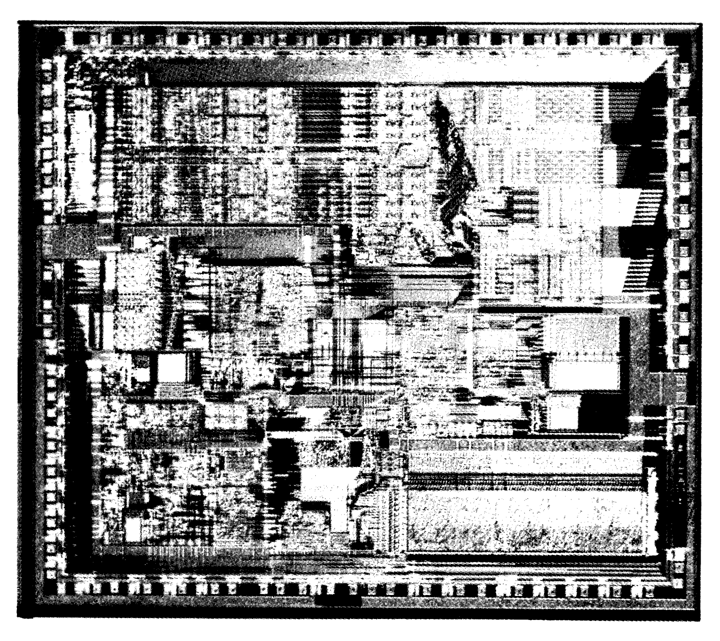
32-Bit-Mikroprozessor der 2. Generation

Sicher ist jeder Hersteller von 32-Bit-Prozessoren davon überzeugt, daß sein Produkt die Anforderungen der Kunden am besten erfüllt, allerdings entscheidet letztendlich der Kunde, welchen Typ er verwenden wird. In dieser Hinsicht konnte die Serie 32000 innerhalb kurzer Zeit einige Pluspunkte erzielen: Siemens, Bosch, Burroughs, Tektronix, Intergraph (zweitgrößter CAD/CAM-Hersteller nach IBM) und Daisy wählten diese Familie, wenn es galt, die Leistung ihrer bestehenden Systeme zu erhöhen oder eine Maschine zu entwerfen, die in den Leistungsbereich einer VAX vordringt, aber nur einen Bruchteil davon kostet und sich auf dem Schreibtisch des Ingenieurs unterbringen läßt.

Die Entwicklung der Serie 32000 geschah unter ähnlichen Gesichtspunkten wie die eines LAN- oder Hard-Disk-Controller-Chipsatzes. Es sind Investitionen in einer Höhe erforderlich, daß es ein Hersteller sich nicht leisten kann, einen falschen Weg zu beschreiten. Nur ein enges Zusammenarbeiten zwischen Kunden und Halbleiterherstellern garantiert, daß die Forderungen beider Seiten erfüllt werden, nämlich für den Kunden das gewünschte Produkt zum richtigen Preis und für den Hersteller ein lohnender Absatzmarkt.

Als National Semiconductor mit potentiellen Mikroprozessorabnehmern über deren wichtigste Anforderungen diskutierte, kristallisierten sich fünf Hauptpunkte heraus. Der erste ist die Unterstützung höherer Programmiersprachen, d. h. die Programmierung in C oder Pascal, wobei möglichst verdichteter Code erzeugt wird und ein Ingenieur keine Zeit zur Optimierung in Handarbeit verliert. Das zweite ist die effiziente Verwaltung großer Speicherbereiche, denn Mikroprozessoren kommen heute mit 64 KByte Speicher nicht mehr aus, in der Regel sind mehrere MByte erforderlich. Drittens besteht ein Bedarf an effizienter Unterstützung von Fließkommaoperationen, insbesondere weil von 32-Bit-Mikroprozessoren ein Bereich abgedeckt wird, der sehr rechenintensive Aufgaben stellt. Als viertes besteht der Wunsch, alle Elemente eines Mikrocomputers frühzeitig verfügbar zu haben, d. h. daß die vollständige Systemleistung in Silizium erhältlich sein muß, daß sich ein Kunde lediglich auf seine Anwendung konzentrieren kann. Fünftens muß die Architektur auch langfristigen Ansprüchen genügen, damit die hohen Investitionen des Kunden sich auch auszahlen können.

Die ersten Chipsätze der Serie 32000 wurden bereits unter Berücksichtigung dieser Gesichtspunkte konzipiert. Heute stehen drei CPUs zur Verfügung, die vollständig softwarekompatibel sind und sich an den Bedürfnissen der 8-, 16- und 32-Bit-Märkten orientieren. Dies sind z. B. CMOS-Ausführungen mit erweitertem Temperaturbereich, Vorauswahl für militärische Anwendungen oder strahlungsfeste Elemente, die sich derzeit in Entwicklung befinden. Die 32000-Familie kann daher mit ihrer wirtschaftlichen Preisgestaltung praktisch alle Ansprüche des Marktes erfüllen. Darüber hinaus sind z. B. der MMU-Baustein NS32082, der die virtuelle Speicherverwaltung nach dem Demand-Paged-



Bereits Realität: 32-Bit-Mikroprozessor der 2. Generation. Auf dem Chip des 32332 befinden sich etwa 90 000 Transistorfunktionen

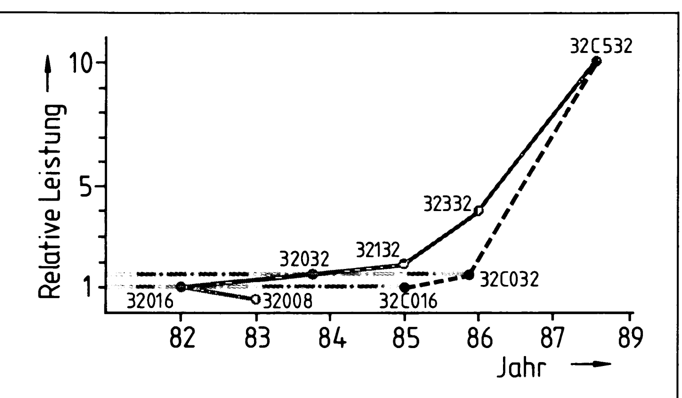
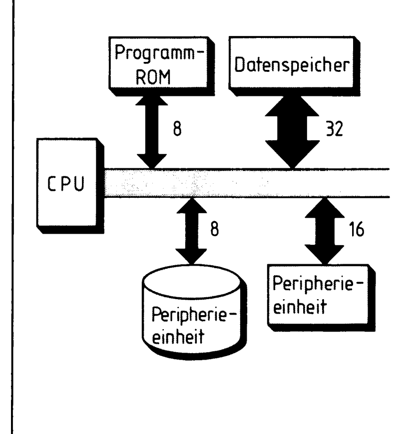


Bild 1. Die Richtung der Weiterentwicklung für die Bauelemente der Serie 32000 bis zum Jahre 1987/88 zeigt die Leistungsverbesserung bei der Implementierung der Architektur, ohne daß dabei das Benutzer-Interface verändert werden muß. Softwarekompatibilität ist dazu der Schlüsselfaktor. Es ist auch daran gedacht, Produkte der Serie 32000 in der Standard-Zellen-Bibliothek anzubieten

zielle ALU mit einem Barrel-Shifter, der Adreß- und Index-Berechnung beschleunigt, eine erweiterte Queue von 20 Byte, ein verbessertes Bus-Zyklus-Timing, das es erlaubt, bei 15 MHz mit einem Cache-Speicher ohne Einfügen von Wartezuständen zu arbeiten (dies scheint eine Minimalforderung zu sein, ist aber z. B. beim 68020 nicht möglich), eine dynamisch konfigurierbare Busbreite von 8, 16 oder 32 Bit in Abhängigkeit des adressierten Speicher- oder E/A-Bereiches (*Bild 2*), eine Burst-Betriebsart für einen Speicherzugriff mit zwei Zyklen, durch den Transfer zur Queue oder zu Slave-Prozessoren beschleunigt werden, ein verbessertes Slave-Interface mit 16- oder 32-Bit-Datenpfad sowie zwei separate leistungsfähige Bus-Fehler- und -Abbruch-Protokolle mit Befehls-Neustart. Dies alles führt zur 3fachen Leistung im Vergleich zum 32032,



◁ Bild 2. Die dynamisch veränderbare Busbreite ermöglicht eine sehr kosteneffektive Systemkonfiguration. So ist beispielsweise lediglich ein EPROM-Baustein für den Bootstrap-Vorgang notwendig

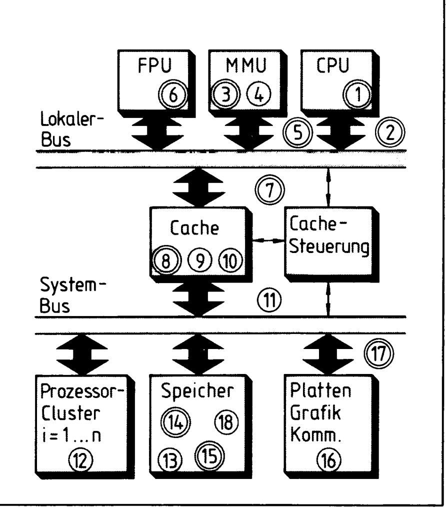


Bild 3. ▷ Einige der Faktoren, die bei der Systemsimulation berücksichtigt wurden, um die Verbesserungen des NS32332 zu definieren: 1. der CPU-Anteil, 2. Bus-Protokoll, 3. MMU-Trefferrate, 4. MMU-Miss-Overhead, 5. FPU-Protokoll, 6. FPU-Leistung, 7. Belegung des lokalen Bus, 8. Cache-Zugriffszeit, 9. Cache-Miss-Overhead, 10. Cache-Belegung, 11. System-Bus-Aktivität, 12. andere Aktivitäten des Prozessors, 13. Speicher-Zugriffs-Zeit, 14. Speicher-Refresh, 15. Speicher-ECC, 16. Leistung der Laufwerke oder andere Peripherieeinheiten, 17. DMA, 18. Zusammensetzung der Befehle

Verfahren vornimmt sowie die FPU NS32081, die ein außerordentlich interessantes Preis-Leistungs-Verhältnis bietet, bisher einmalig auf dem Markt.

In diesem Beitrag wird die 2. Generation der 32-Bit-Computerelemente vorgestellt, die bereits jetzt als Muster verfügbar ist. Es handelt sich hierbei nicht nur um einen weiteren Schritt in der Produktentwicklung bei National Semiconductor, die CPU 32332 wird in bezug auf die Mikroprozessorleistung neue Standards setzen. Außerdem ist das Unternehmen derzeit dabei, die Spezifikationen der 3. Generation, die aus dem Typ 32C532 besteht, festzulegen. Diese CPU ist für 1987 geplant (*Bild 1*). Die Erfahrungen mit 32-Bit-Konzepten wurden genutzt, um mit dem 32332 einen „Super-Prozessor" zu entwickeln, der die gleiche Architektur wie die existierenden 32000-Prozessoren bietet, allerdings bei vollständiger Softwarekompatibilität eine Leistungssteigerung in bezug auf den Typ 32032 um den Faktor 3 erreicht.

Die wichtigsten Merkmale des Typs 32332, die über diejenigen des 32032 hinausgehen, sind der 32 Bit breite Adreßbus (4 GByte linearer Speicherbereich), eine spe- wobei der Chip im kostengünstigen 84poligen Pin-Grid-Gehäuse untergebracht werden konnte und vollständige Software-Kompatibilität beibehalten wird.

Systemleistung entscheidend

Bei 32-Bit-Mikroprozessoren ist die reine Leistung der CPU natürlich in erster Linie wichtig, allerdings ist es wesentlich kritischer, wie die CPU sich verhält, wenn sie in einer typischen 32-Bit-Systemumgebung arbeiten muß. Die Leistungsmerkmale in einem System sind das Produkt aus CPU- und System-Leistung, wobei das Zusammenspiel zwischen CPU-Funktionseinheiten und externer Welt berücksichtigt werden muß. *Bild 3* zeigt eine typische 32-Bit-Systemorganisation, die verschiedenen Faktoren, die die Leistung beeinflussen, und diejenigen, die bei der Konzipierung des NS32332 berücksichtigt wurden.

Der Kern eines Computers, der hauptsächlich um eine CPU und die eng damit verbundenen Peripherieeinheiten wie Speicherverwaltungseinheit und Fließkomma-

einheit angeordnet sind, werden über einen lokalen Bus mit einem Cache- oder lokalen Speicher verbunden. Ein solcher lokaler Bus muß schnelle Kommunikation erlauben und nutzt die Mikroprozessorleistung voll aus. Zugriff auf den lokalen Speicher darf nur ohne Wartezyklen oder Arbitrations-Verzögerungen erfolgen. Dieser lokale Bus ist üblicherweise sehr eng mit dem Mikroprozessorbus verkoppelt und für sehr schnelle Arbeitsweise ausgelegt. Ein Mikroprozessor verwendet den größten Teil seiner Zeit darauf, auf den Speicher zuzugreifen, so daß die optimale Auslegung dieses Interfaces ein kritischer Punkt in bezug auf hohe Systemleistung ist.

Einen Systembus kann man heute treffender als „globalen Bus" bezeichnen, weil er hauptsächlich zum Verbinden verschiedener Prozessoren oder zum Zugriff auf globale Peripherieeinheiten dient. In allen Mikroprozessoren, die einen gemeinsamen Bus verwenden, gibt es immer wieder Verzögerungen aufgrund der Arbitration, weil zu einem bestimmten Zeitpunkt nur immer ein Master den Bus steuern kann. Bei 8- oder 16-Bit-Prozessoren führte dies nicht unbedingt zu einer Leistungsminderung, weil hier so hohe Ansprüche nicht gestellt werden. Der begrenzende Faktor liegt in dieser Leistungsklasse nicht beim Bus. Bei 32-Bit-Mikroprozessoren ist die nachteilige Wirkung der Arbitrationsverzögerungen beim Speicherzugriff nicht mehr tragbar. Konsequenterweise verwendet man den lokalen Speicher zum Ablegen der meisten Befehle und Daten, die der dazugehörige Prozessor benötigt. Auf den lokalen Speicher kann dieser „privat", ohne Geschwindigkeitsbegrenzungen zugreifen.

Der erste Faktor, der die Systemleistung beeinflußt, ist das Busprotokoll. In diesem Punkt konnten beim 32332 Verbesserungen erreicht werden, so daß sich ein klarer Vorteil gegenüber beispielsweise dem Typ 68020 ergibt. Diese Verbesserungen liegen in erster Linie in der Möglichkeit, auf normale dynamische Speicher (lokaler oder Cache-Speicher) ohne Wartezyklen auch bei 15 MHz zuzugreifen, zweitens in einem Zyklus von nur vier Takten auch mit der MMU (beim 68020 sind fünf Takte in einem Zyklus mit der MMU erforderlich), drittens in der Burst-Betriebsart, die konsekutive Transfers in nur zwei Taktzyklen ermöglicht und letztendlich in einer Busfehler-Reaktivierungseinrichtung, die eine separate Leitung und ein Befehls-Restart-Verfahren verwendet, das schneller arbeitet und leichter zu verwenden ist, als dies bei der vergleichbaren Einrichtung des 68020 möglich ist.

In bezug auf das Slave-Protokoll konnten Verbesserungen erreicht werden, weil der Datenpfad für die neuen Slave-Peripherieeinheiten auf 32 Bit erweitert wurde. Dies ermöglicht schnellere Ausführungsgeschwindigkeit aller Befehle, die mit dem Slave-Betrieb zu tun haben, weil effektivere Datentransfers zwischen Slave und CPU stattfinden können. Darüber hinaus sollen neue Slave-Einheiten vorgestellt werden, die verbesserte Merkmale bieten. (Diese werden in einem weiteren Beitrag näher vorgestellt.)

Der CPU-Beitrag zur Systemleistung

Der Beitrag der CPU an der Systemleistung läßt sich durch die Ausführungsgeschwindigkeit und die internen Verbesserungen der architektonischen Implementierungen in der 32000-Familie messen. Wie bereits gesagt, wurden an der Architektur selbst keine Veränderungen vorgenommen, weil diese Familie eine sehr einfache, saubere und leistungsfähige Struktur besitzt. Was man gut versteht, kann man auch sehr einfach implementieren. Eine Architektur muß nicht komplex sein, um eine große Leistung zu bieten.

Beim NS32332 findet man eine Anzahl von Verbesserungen, die dafür sorgen, daß die Anzahl der Takte pro Befehl verringert ist. Dazu zählt z. B. ein separater schneller Addierer für die Adreßberechnung, ein Barrel-Shifter für ein, zwei oder drei Verschiebungen, der insbesondere für Adreß-Berechnungen geeignet ist (*Bild* 4), ein neuartiger Mikrocode und ein verbessertes Businterface, das normale Buszyklen in vier Takten auch in Zusammenhang mit der MMU abarbeitet und das die Bursts für Nibble-Mode-Speicher in nur zwei Takten pro Transfer überträgt. Darüber hinaus ist die CPU NS32332 für 15 MHz spezifiziert.

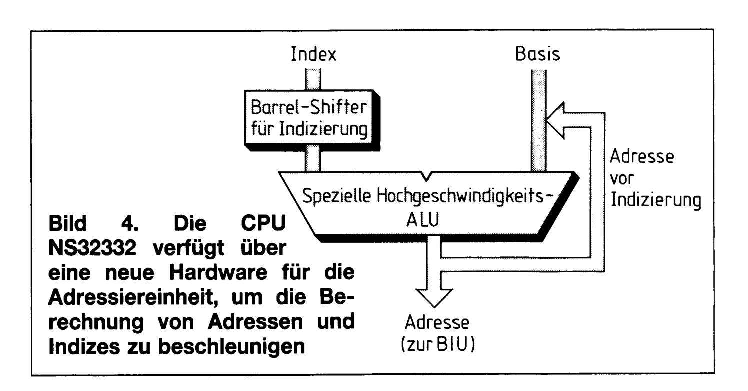


Bild 4. Die CPU NS32332 verfügt über eine neue Hardware für die Adressiereinheit, um die Berechnung von Adressen und Indizes zu beschleunigen

50% der Leistungsverbesserungen der CPU 32332 stammt aus dem schnellen Addierer, der Verbesserung des Mikrocodes und anderen Merkmalen, die den Code beschleunigen. Das neue Businterface und die vergrößerte Queue (von 12...20 Byte) erhöhen die Geschwindigkeit um etwa 30%. Wenn man dies mit der 50%igen Taktbeschleunigung von 10 auf 15 MHz kombiniert, erhält man einen Gesamtwert von etwa 3 (1,5 × 1,3 × 1,5 = 2,93). Die Geschwindigkeitsverbesserung in bezug auf den 32032 läßt sich nachmessen.

Ein sehr wichtiger Punkt ist die Tatsache, daß zur Ausnutzung der hohen Systemgeschwindigkeit bei einem Takt von 15 MHz der Zugriff vom Mikroprozessor auf den Speicher ohne Wartezyklen möglich sein muß. Der zeitliche Ablauf des Speicherzugriffs wird später detailliert beschrieben. Normale dynamische Speicher (100 oder 150 ns) lassen sich an die CPU 32332 ohne Wartezyklen anschließen, weil das Ready-Signal relativ spät abgefragt wird. Beim 68020 ist dies nicht der Fall, denn die Zeit, in der ein Cache- oder lokaler Speicher

antworten könnte, ist nicht lang genug. Dieser Gesichtspunkt ist sehr wichtig, weil üblicherweise Benchmarks nur unter der Voraussetzung gelten, daß keine Wartezyklen ablaufen. In der Praxis ergeben sich daher erhebliche Unterschiede.

Die unterschiedlichen Verbesserungen der Implementierung des 32332 beeinflussen nicht die Beschaffenheit der Benutzerschnittstelle zum Mikroprozessor. Sowohl der Befehlssatz als auch die Adressierungsarten sind unverändert. Bei 3facher Leistung in bezug auf den Typ 32032 wurde vollständige Softwarekompatiblität beibehalten. Dieser Gesichtspunkt spielt eine besondere Rolle, weil ein Anwender die Investitionen schützen muß, die im oberen Leistungsbereich sehr großen Umfang annehmen können.

Eine neue Adreßeinheit, die aus einer speziellen Hochgeschwindigkeits-ALU und einem Barrel-Shifter besteht, wurde aus zwei Gründen hinzugefügt: Erstens sollten die internen Transferzeiten verkürzt werden, denn die zweite ALU erlaubt Parallelbetrieb (gleichzeitige Berechnung von arithmetischen und Adreß-Werten) und zweitens sollte mit Hilfe der Barrel-Shifter-Funktion die Indizierung beschleunigt werden. Skalierte Index-Adreß-Modifizierer werden sehr häufig von Compilern verwendet, wenn diese Code für die Serie 32000 erzeugen. Ein Beispiel für die merkliche Verbesserung zeigt folgender Befehl:

MOVD R0 [R1:D] , R2

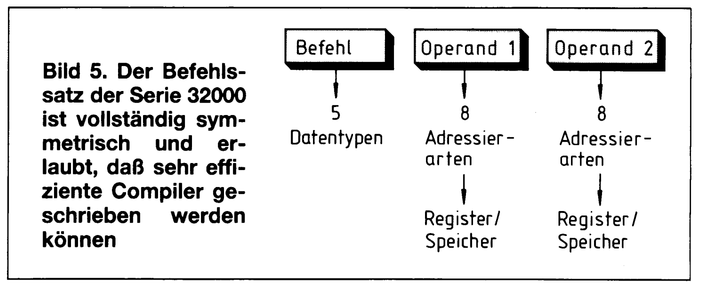
Dieser Befehl dient üblicherweise dem Transfer eines Doppelworts, das sich in einer Tabelle mit der Startadresse im Register R0 befindet, in das Register R2. Die Elementezahl N in dieser Tabelle befindet sich wiederum im Register R1. Beim CPU-Typ 32032 benötigt dieser Befehl 21 Taktzyklen, während beim Typ 32332 lediglich elf Taktzyklen erforderlich sind.

Compiler-Effizienz durch symmetrischen Befehlssatz

Der Befehlssatz des 32332 ist mit demjenigen des 32032 identisch. Dieser wurde insbesondere zur effizienten Unterstützung höherer Programmiersprachen ausgelegt. Das bedeutet, daß nicht nur Operationen auf höherer Ebene berücksichtigt wurden, sondern auch entsprechende Datenstrukturen, z. B. Arrays, gepackte Arrays, Records, Zeichenketten, Bitfelder oder Semaphoren. Ein weiterer wichtiger Vorteil des Befehlssatzes der Serie 32000 im Vergleich zu dessen Mitbewerbern (einschließlich 68020) ist die vollständige Symmetrie (*Bild* 5). Jeder Softwareingenieur oder Compilerentwickler wird sicherlich auch der Meinung sein, daß beispielsweise die Adressierstruktur des 68020 sehr komplex und unregelmäßig ist.

Beim 32000 hat jede Instruktion zwei echte Operanden, die unabhängig voneinander im Speicher oder Register sein können und entweder einem Byte, Wort oder Doppelwort bestehen. Darüber hinaus läßt sich auf jeden Operanden mit Hilfe jeder der möglichen Adres-

Bild 5. Der Befehlssatz der Serie 32000 ist vollständig symmetrisch und erlaubt, daß sehr effiziente Compiler geschrieben werden können



sierarten zugreifen. Es gibt keine Ausnahme, so daß es wesentlich leichter ist, optimierte Compiler zu schreiben. Nachdem die Serie 32000 drei Jahre auf dem Markt ist, benutzen eigentlich alle Softwareingenieure optimierte Compiler für den 32000, wobei sich eine bemerkenswerte hohe Code-Dichte ergibt.

Als Beispiel für die Überlegenheit der 32000-Familie soll hier die Verwaltung von Arrays gezeigt werden. Der Typ 68020 verfügt über einen neuen Befehl, der es festzustellen erlaubt, ob eine Quantität innerhalb von zwei Grenzwerten liegt. Dieses Wertepaar befindet sich in einem Speicherplatz, der durch allgemeine Adressen angegeben wird. Falls die Menge nicht innerhalb der Grenzen liegt, spricht eine Trap-Funktion an. Beim 32000 bietet sich der Vorteil, daß sowohl der Test ausgeführt wird als auch Zusatzfunktionen, nämlich die Plazierung der getesteten Menge in ein Register, wenn sie unter dem unteren Grenzwert liegt (Normieren des Indexwertes) wo es als ein Index für den Array Verwendung finden kann. Der Prüfbefehl des 32000 erzeugt nicht automatisch eine Trap. Er setzt ein Flag, das sich sehr schnell mit einem 1-Byte-Befehl („Test-Flag") prüfen läßt. Dies ist auf jeden Fall schneller als die generelle Trap-Funktion. Darüber hinaus und gleichzeitig als Beispiel für die Durchgängigkeit des 32000-Befehlssatzes berechnet ein Index-Befehl die Plazierung eines Elementes in einem zweidimensionalen Array aus dem Wert von zwei normierten Indizes und dem maximalen Wert des ersten Index. Dieser Index-Befehl benutzt normierte Indexwerte, die von dem Befehl „Check" angeliefert werden. Für ein n-dimensionales Array muß der Befehl „Index" (n-1) mal ausgeführt werden, um auf ein Element in diesem Array zuzugreifen. Außerdem berechnet der Index-Befehl den Rang des Elementes in der Linear-Darstellung des Arrays im Speicher, ohne daß dabei zu berücksichtigen ist, ob es sich um Byte-, Wort- oder Doppel-Wort-Elemente handelt. Der skalierte Index-Adreßmodifizierer wird von dem Befehl benutzt, der die Operation mit dem Element des Arrays ausführt. Dies ergibt ein durchgängiges Konzept, das einfach, logisch und effizienter für einen Compiler und besser lesbar für den Benutzer ist.

Adressierungsarten

Die Adressierarten, die für alle CPUs der Serie 32000 zur Verfügung stehen, sollten mit der komplexen Struktur des 68020 einmal verglichen werden. An den Ergeb-

nissen zeigen sich die Unterschiede. Um einen Vergleich durchführen zu können, ist es sehr hilfreich, eine einheitliche Methode zur Beschreibung der Adressierungsarten sowie Tabellen mit den verfügbaren Möglichkeiten der beiden CPUs zu haben. Für den Vergleich sollen daher die Schreibweisen der Sprache C verwendet werden. Es seien:

- „D“ jedes beliebige Universal-Register der 32000-CPU oder Daten-Register des Typs MC68020.
- „A“ jedes beliebige Speicher-Register der 32000-CPU (Sp, Fp, Bp) oder jedes der Adreß-Register des MC68020.
- „R“ jedes Daten- oder Adreß-Register des 68020 (D oder A).
- „Pc“ der Programmzähler,
- „d“ ein Displacement und
- „i“ ein Index (z. B. ein weiteres Displacement).

Die Adressierarten der CPU 68020 lassen sich dann folgendermaßen beschreiben:

(0)	D		
(1)	A		
(2)	*A		
(3)	*(A++)		
(4)	*(−−A)		
(5)	*(A + d)		d ist ein Wort
(6)	(a)	*(A + D + d)	d ist ein Byte
	(b)	*(*(A + R + d) + i)	d,i sind Null, Wort oder Doppelwort
	(c)	*(*(A + d) + (R + i))	d,i sind Null, Wort oder Doppelwort
(7)	(0)	absolute	d ist Wort
	(1)	absolute	d ist Doppelwort
	(2)	*(Pc + d)	d ist Wort
	(3)	(a)	
		(b) > Wie (6), aber mit Pc anstelle von A	
	(4)	Immediate	

Die Adressierungsstruktur der Serie 32000 ist wesentlich gleichmäßiger und einfacher:

(0–7)	D	
(8–15)	*(D + d)	d ist Byte, Wort oder Doppelwort
(24–26)	*(A + d)	d ist Byte, Wort oder Doppelwort
(27)	*(Pc + d)	d ist Byte, Wort oder Doppelwort
(16–18)	*(* (A + d) + i)	d,i sind Bytes, Worte oder Doppelworte
(19)	— reserviert —	
(20)	Immediate	
(21)	Absolute	
(22)	*(E(n) + d)	Externe Betriebsart d ist Byte, Wort oder Doppelwort
(23)	*(Sp++) oder *(Sp−−)	Top of Stack
(28–31)	*(&irgendetwas + d)	d ist Byte, Wort oder Doppelwort

Um die zwei Adressierarten für einen typischen Compiler-Zugriff auf Datenstrukturen höherer Ebene vergleichen zu können, soll vorausgesetzt werden, daß A6 im 68020 mit dem Fp (Frame-Pointer) des NS32000 übereinstimmt und daß die 68020-Register Di mit den 32000-Registern Ri übereinstimmen. Für die 68020-Adressen wird die C-Schreibweise gewählt.

Beispiel 1
Hole ein Wort aus einem Array von Strukturen über eine Pointer im laufenden Frame
Beim 68020 benötigt der Befehl:

Mov.w * (*(A6 + d) + (Do + i)), D1

Für den Op-Code 2 Byte, die Erweiterung 2 Byte, das D-Displacement 2 Byte, das i-Displacement 2 Byte, also insgesamt 8 Byte.
Beim 32032 oder 32332 benötigt der Befehl:

Movw i (d(Fp)) [R0 : b], R1

Für den Op-Code 2 Byte, das Index-Byte 1 Byte, das D-Displacement 1 Byte, das i-Displacement 1 Byte, also insgesamt 5 Byte.

Beispiel 2
Holen von einer Struktur über einen Pointer in einem Register
Im 68020 benötigt der Befehl

Movw i (d(Fp)) [R0 : b], R1

Für den Op-Code 2 Byte, für das Displacement 2 Byte, also insgesamt 4 Byte.
Beim 32000 benötigt der Befehl

Movw i (d(Fp)) [R0 : b], R1

Für den Op-Code 2 Byte, für das Displacement 1 Byte, also insgesamt 4 Byte.

Diese Beispiele stellen einen einfachen Vergleich der beiden Adressierarten dar, bei denen Byte-Displacements vorausgesetzt werden. In der Praxis können die meisten Displacements in einem Programm mit Hilfe eines Bytes ausgedrückt werden. Dies trifft darüber hinaus für jedes Displacement bei jeder beliebigen Adressierart zu. Außerdem ist der große Vorteil eines echten 2-Adreß-Befehlssatzes der Serie 32000 hier nicht in Erwägung gezogen, weil als Ziel ein Register dient.

Verbessertes Busprotokoll

Der wichtigste Faktor, der die Systemleistung bestimmt, ist das Bus-Protokoll. Es handelt sich hierbei um die Haupt-Kommunikationseinrichtung zwischen CPU und lokalem Systemspeicher. Das verzögerte Abfra-

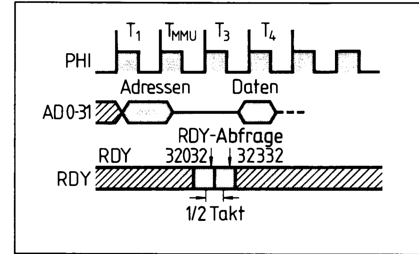


Bild 6. Die späte Abfrage des Ready-Signals beim 32332 ermöglicht die externe Cache-Implementierung. Damit läßt sich auf den Cache ohne Wartezyklus zugreifen

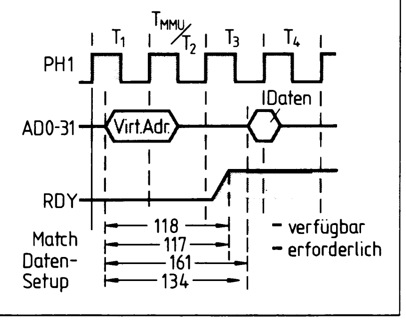


Bild 7. Das Cache-Interface des 32332 ist so ausgelegt, daß es Cache-Speicher in realistischen Systemen ansteuern kann, auch bei 15 MHz

gen des Ready-Signals (*Bild* 6) gibt dem Speicher mehr Zeit zu antworten und erlaubt die Verwendung von standardmäßigen preisgünstigen Speicherchips. Wenn man berücksichtigt, daß ein Mikroprozessor mehr als 90 % der Zeit mit seinem lokalen Speicher kommuniziert, bedeutet dieses Merkmal einen großen Vorteil des 32332 im Vergleich zum 68020.

Bild 4 zeigt den Unterschied zwischen dem Typ 32332 und 68020. Diese Unterschiede lassen erkennen, daß der 68020 offensichtlich ein Chip-Konzept darstellt, während der 32332 unter Berücksichtigung des System-Gedankens konzipiert wurde, weil er so ausgelegt ist, daß er Cache-Speicher in realistischen Systemen ansteuern kann. Die Zykluslänge für den Typ 32332 beträgt bei 15 MHz 264 ns, wobei ein virtuelles System die MMU benutzt. Beim 68020 liegt die Zykluslänge von 360 ns bei 16,7 MHz etwa um 40 % langsamer.

Vom 68020 ist bekannt, daß er in der Lage ist, Bustransfers in drei Taktzyklen auszuführen – ein Wert, der offensichtlich vorteilhaft gegenüber den vier Zyklen des 32332 ist. Wie macht sich das in einer echten Systemimplementierung bemerkbar? Das DSACK-Signal des 68020 läßt sich mit S2 anlegen, und wenn dies fertig ist, schließt der Zyklus bei S5 ab. („S" sind die Taktperioden oder Halb-Takt-Zyklen, beginnend bei S0.) Wenn das DSACK-Signal bis S3 verzögert ist (ein halber Takt), dauert der Buszyklus vier Zyklen und endet bei S7. Wenn DSACK während der ersten Hälfte eines Taktzyklus anliegt, schließt üblicherweise der Buszyklus am Ende des folgenden Takts ab. Wenn DSACK in der zweiten Hälfte eines Takts anliegt, endet der Buszyklus am Schluß des zweiten darauf folgenden Taktes.

An den Adreßleitungen des 68020 liegt während S0 der gültige Wert an, so daß zum Erreichen eines Drei-Takt-Zyklus DSACK innerhalb von zwei Perioden reagieren muß. Dies bedeutet bei einer Einstellzeit von 5 ns 95 ns bei 10 MHz und 55 ns bei 16,6 MHz.

Solch kurze Zeiten lassen sich auch von einem Cache-Speicher nicht erreichen. Der schnellste, derzeit als Muster erhältliche, Cache-Komparator in kompatibler Technologie hat eine Match-Zeit von 117 ns. Bei einem 68020-System mit derzeitiger Technologie und einem Cache-Speicher, der einen Durchschreib-Mechanismus benutzt, ist zu erwarten, daß er für einen Off-Chip-Lesezyklus auf der Platine fünf Taktzyklen braucht, sieben Taktzyklen für einen Lesevorgang außerhalb der Platine und acht Taktzyklen für einen Schreibzyklus.

Beim 32332 liegen auf den Adreßleitungen die gültigen Werte z. Zt. T1,5, d. h. zur fallenden Flanke von Phi 1 innerhalb von T1. RDY (das Äquivalent zu DSACK) muß zum Zeitpunkt T3,5 anliegen, dies ist (unter Berücksichtigung von 5 ns Einstellzeit) bei 10 MHz innerhalb von 195 ns und bei 15 MHz innerhalb von 118 ns. Die CPU eines 32332 ist damit in der Lage, auf externe Cache-Speicher ohne Wartezyklen zuzugreifen, während der Typ 68020 dies nicht kann.

Externer Cache oder interner Befehls-Cache?

Die Unterstützung des externen Cache-Speichers erreicht man durch verbessertes Zeitverhalten der Schreib-/Lese-Vorgänge und Burst-Transfers. Kurz gesagt: Beim 32332-Konzept mit der 20-Byte-Queue, den richtigen Zeitverhältnissen zum Zugriff auf externe Cache-Speicher und der Burst-Betriebsart ergibt sich eine bessere Leistung im Vergleich zu einem intern limitierten Cache-Speicher (z. B. 256 Byte), der lediglich für Befehle vorgesehen ist (68020). Wie bisher bekannt ist, soll die CPU 80386 von Intel, von der man zunächst annahm, daß sie einen internen Cache besitzt, die Unterstützungsmöglichkeit für externe Cache-Speicher bieten. Wenn es die Technologie ermöglicht, einen effizient arbeitenden Befehls- und Daten-Cache (minimal 8 KByte) mit der gesamten dazugehörigen Logik zu integrieren, wird man diese auch implementieren. Zum derzeitigen technischen Stand ist dies nicht möglich.

Die Burst-Betriebsart (*Bild* 8), die vom 32332 unterstützt wird, nutzt ineinander verschachtelte Speicher und RAMs mit Nibble- und statischem Spalten-Zugriff

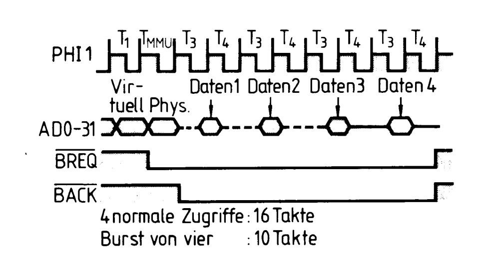


Bild 8. Die Burst-Betriebsart erlaubt 60 % schnellere Transfers. Diese wird hauptsächlich zum Prefetch von Instruktionen in die Queue benutzt, oder zum Holen von nonaligned Operanden, dem Transfer von Operanden zu Slave-Prozessoren oder dem Transfer von 32-Bit-Daten auf 8- oder 16-Bit-Bussen

aus. Dies beschleunigt Transfers über geringe Entfernungen, wie z. B. beim Auffüllen der Queue, Zugriff auf sogenannte „non-alignet" Operanden, Transferieren von Fließkomma-Operanden oder Ausführen von 32-Bit-Transfers auf einem 8- oder 16-Bit-Bus (unter Verwendung der dynamischen Konfigurierbarkeit des 32332-Datenbus). Es ergibt sich kein Overhead bei Zugriffen, die nicht im Burst-Betrieb erfolgen. Die Burst-Betriebsart wird außerdem über den Systembus unterstützt. Dieses Merkmal ermöglicht den Transfer von 16 aufeinander folgenden Bytes in zehn Taktzyklen anstatt in 16.

Die CPU 68020 verfügt über einen Cache-Speicher von 64 Doppelworten auf dem Chip, der nur für Befehle ausgelegt ist. Wenn man davon ausgeht, daß 60 % der Buszugriffe für Befehls-Fetches erfolgen und die Trefferrate für den Cache 75 % beträgt, ergibt sich eine Reduzierung der Busaktivität von 45 %. Wenn man weiter davon ausgeht, daß ein Cache-Treffer die Ausführungszeit um 50 % verringert, kommt man auf eine Erhöhung der Prozessorgeschwindigkeit um 22 %. Die CPU 32332 besitzt auf dem Chip keinen Cache-Speicher. Allerdings sollte die hohe Geschwindigkeit beim Zugriff auf Speicher in Burst-Betriebsart bei richtiger Koordinierung mit der Befehls-Queue bessere Eigenschaften in Zusammenhang mit einem passenden Cache-Speicher außerhalb des Chips erreicht werden können. Die Größe dieser Queue (20 Byte) wurde so gewählt, daß es möglich ist, den nächsten Befehl darin noch unterzubringen (ein Mittelwert: 4 Byte) und 16 weitere Bytes zum Auslösen eines Burst-Prefetch (*Bild 9*).

Dieser Punkt läßt sich sehr einfach in einer Tabelle zusammenfassen, die die Eigenschaften der beiden Chips miteinander vergleicht:

Konzept	In-Line-Code	Verzweigung
Cache (68020)	0,8 Takte	8 Takte
Queue (32332)	0 Takte	11,0 Takte

Diese Zahlen werden im folgenden abgeleitet, wobei sich einige Auswirkungen auf die tatsächliche Leistung ergeben.

Bei der CPU 68020 handelt es sich um eine Pipeline-Maschine, bei der es bei Verzweigungen zu Nachteilen kommt. Die Pipe besteht aus drei Stufen, die alle gefüllt sein müssen, damit die Maschine arbeiten kann. Daher muß nach einer Verzweigung die CPU 68020 zwei Doppel-Wort-Fetches ausführen, um die Pipe neu zu füllen, bevor die Maschine wieder mit der Programmausführung beginnen kann. Diese nachteilige Pause ergibt sich auch dann, wenn die Verzweigung zu einem Speicherplatz im Cache erfolgt. Sie beträgt zwei Fetch-Perioden vom Cache, d. h. jeweils zwei Zyklen, also insgesamt vier Zyklen. Wenn die Verzweigung zu einem Speicherplatz erfolgt, der nicht im Cache ist, beträgt die Verzögerung zwei Fetch-Vorgänge zu einem Speicherplatz außerhalb des Chips, d. h. sechs Takte (theoretisches Minimum) oder zehn Takte (praktisches Minimum). Der zeitliche Nachteil bei der Verzweigung liegt zwischen 4 und etwa 14 Taktzyklen, wobei das Mittel von dem Verhältnis der Verzweigungsziele, die im Cache liegen, gewichtet wird. Wenn man einen Wert von 60 % bei den Verzweigungen zum Cache annimmt, liegt die mittlere Verzögerung bei etwa acht Zyklen. 60 % sind sehr praxisnah, weil ein Direct-Mapped-Cache eine geringe Trefferrate zeigt.

Bei der CPU 32332 sind mehr Stufen in einer Pipe, und diese müssen nicht alle zur gleichen Zeit gefüllt sein. Die Verzweigungsverzögerung ist die Summe der internen Verzögerungen (bei der Verarbeitung der Verzweigung) und die Zeit, die zum Aufnehmen der Operanden erforderlich ist. Operanden sind im Mittel 3 Byte lang. Die Art der Berechnung ist aufgrund der Burst-Verarbeitung ein wenig kompliziert. Wenn man einen Burst auslöst, muß die CPU 32332 diesen beenden, allerdings beginnt die Ausführung eines Befehls, sobald die Operanden sich in der Queue befinden. In der Praxis

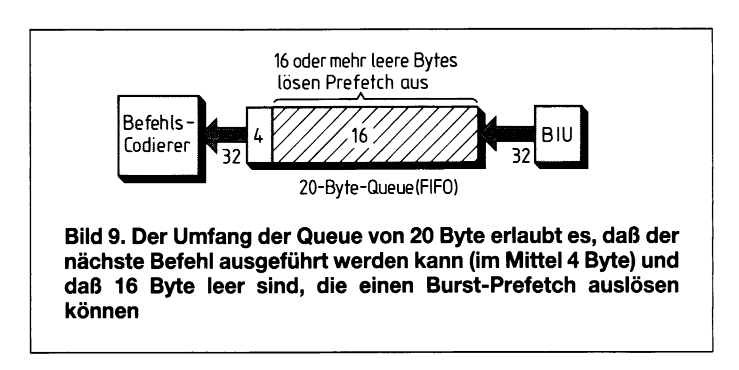


Bild 9. Der Umfang der Queue von 20 Byte erlaubt es, daß der nächste Befehl ausgeführt werden kann (im Mittel 4 Byte) und daß 16 Byte leer sind, die einen Burst-Prefetch auslösen können

liegt das Problem bei der Aufnahme eines Doppel-Wortes, das bei jeder Adresse starten kann. Die Burst-Transfers beginnen von der Startadresse und gehen bis zu einer Grenze, die 16-Byte-Mehrfachadressen entsprechen kann. In der Praxis hat die Startadresse eine gleichmäßig verteilte Chance, daß sie irgendwo eine der 16 Möglichkeiten darstellt. Diese tragen die Namen P0...P15, wobei die Anzahl der Taktzyklen, die erforderlich ist, um vier aufeinanderfolgende Bytes zu erhalten, folgende ist:

P0	P1	P2	P3	P4	P5	P6	P7	P8
4	4	6	6	4	4	6	6	4

P9	P10	P11	P12	P13	P14	P15
4	6	6	4	4	8	8

Wenn die drei Byte-Daten bei P0, P1, P4, P5, P8, P9, P12, P13 beginnen, lassen sie sich in einem Speicherzugriff aufnehmen. Falls sie z. B. bei P2 beginnen, muß das erste Doppelwort aufgenommen werden, um P2 sowie P3 zu erhalten, ein zweiter Fetch-Vorgang muß mit Hilfe der Burst-Betriebsart ausgeführt werden (zwei zusätzliche Taktzyklen) um das übrige Byte zu erhalten. Das gleiche trifft für P3, P6, P7, P10 und P11 zu. Im Falle von P14 und P15 kann der Burst-Mode wegen der Grenzen

nicht benutzt werden. Hier sind zwei vollständige Zyklen auszuführen. Weil in allen Fällen die gleiche Wahrscheinlichkeit besteht, liegt die mittlere Verzögerungszeit einer Verzweigung bei fünf Taktzyklen, die zum Aufnehmen der Operanden erforderlich sind. Insgesamt ergibt sich dabei mit den sechs Taktzyklen für die eigentliche Verzweigung ein Wert von elf Zyklen.

Nun zum In-Line-Code. Die CPU 68020 nimmt Befehle nach Bedarf auf. Dafür wird dem Bus-Interface eine Anfrage immer dann zugesandt, wenn die Pipe eine weitere „Füllung" benötigt. Wenn eine Instruktion extern aufgenommen werden muß, wartet der Prozessor darauf (zwei Taktzyklen). Bei 60 % Trefferrate ist dies in 40 % der Fälle so. Im Mittel liegt die In-Line-Verzögerung bei etwa 0,8 Takten.

Nachdem beim 32332 die Verzweigungsverzögerung abgelaufen ist, gibt es eine sehr große Wahrscheinlichkeit, daß der erforderliche Befehl sich innerhalb der Queue befindet, so daß man die In-Line-Verzögerung als Nulltakte annehmen kann. Für einen Gesamt-Vergleich geht man davon aus, daß im Mittel der Code über fünf Befehle In-Line läuft und dann auf eine Verzweigung trifft. Die Fetch-Verzögerung der beiden Maschinen beträgt dann:

Konzept	In-Line-	Verzweigung	Gesamt
Cache (68020)	5 × 0,8 +	8	= 12 Takte
Queue (32332)	5 × 0 +	11	= 11 Takte

Das Queue-Konzept erscheint also hier als die bessere Möglichkeit. Darüber hinaus ist das Löschen des Cache beim 68020 sehr langsam.

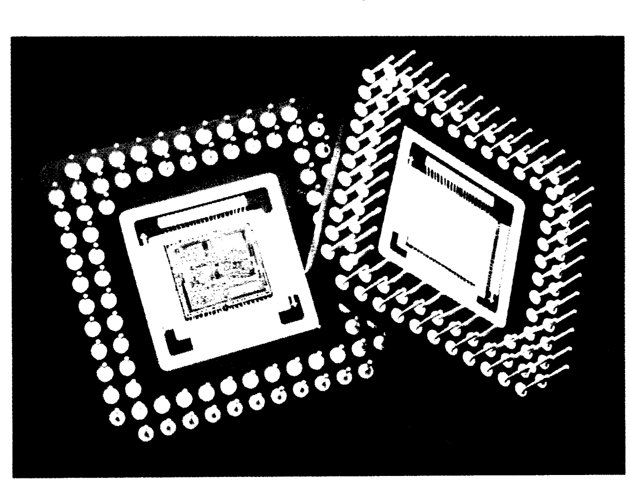
„Wiederbelebung" bei Busfehlern

Wenn beim 68020 ein Busfehler während der Ausführung auftritt, verändern sich so viele einzelne Dinge, insbesondere bei den Autoincrement- und Autodecrement-Adressierarten, daß sich der Befehl nicht mehr mit Hilfe der in den sichtbaren Registern befindlichen Daten erneut starten läßt. Um eine Speicherverwaltung überhaupt zu ermöglichen, reagiert der Motorola-Typ auf einen Busfehler durch genügend interne Zustände, um die Fortführung der Befehlsausführung nach dem Beheben des Busfehlers sicherzustellen (üblicherweise durch Ausführen eines Page-Swap-Vorganges).

Dies wiederum verursacht zwei wichtige Probleme: Erstens ist die Menge der intern zu sichernden Zustände sehr groß: 64 Byte, wenn der Fehler zwischen zwei Befehlen auftritt und 88 Byte, wenn er innerhalb eines Befehls erkannt wurde. Diese müssen zu den 64 Byte der sichtbaren Register addiert werden zuzügl. demjenigen, was an Prozessor-Registern vorhanden ist, so daß sich eine große Menge von Daten ergibt, die im Falle eines Seitenfehlers abzulegen ist. Außerdem garantiert Motorola nicht, daß ein Befehl ohne Softwareintervenierung fortgeführt werden kann. Das Handbuch sagt in diesem Fall folgendes:

32-Bit-Prozessor NS 32332 auf einen Blick

Beim NS32932 handelt es sich um einen Mikroprozessor mit virtueller Speicherverwaltung und einem Adreßbereich von 4 GByte. Die Architektur der 32000-Familie erhielt einige wichtige Erweiterungen, wobei allerdings weiterhin volle Objektcode-kompatibilität bestehen bleibt. Als Unterschied zum bisherigen Konzept hat der Typ 32332 eine Adreßbreite von 32 Bit, einen höheren Befehls-Durchsatz, Cache-Unterstützung und verbesserte Bus



steuerung. Zu den neuen Busfunktionen zählen Bus-Fehler- und Retry-Unterstützung, dynamisch veränderte Busbreitenanpassung, Burst-Zugriff auf Speicher sowie ein verbessertes Slave-Kommunikationsprotokoll. Die höhere Taktfrequenz ergibt beim 32332 im Vergleich zum 32032 eine Leistungssteigerung um den Faktor 2...3. Die CPU 32332 arbeitet mit allen Slave-Prozessoren (16 und 32 Bit) der 32000-Familie.

Wichtigste Merkmale:

- 32-Bit-Architektur
- 4 GByte durchgehender Adreßbereich
- Softwarekompatibilität zur 32000-Familie
- Befehlssatz mit allgemeiner 2-Adreß-Fähigkeit,
  hohem Grad an Symmetrie,
  Adressierarten für höhere Programmiersprachen
- Unterstützung des 16- und 32-Bit-Slave-Prozessor-Protokolls: Speicherverwaltung über NS 32082 oder NS 32C382
  Fließkommaunterstützung über NS 32081 oder NS 32C381
- Erweiterte Busfunktionen:
  Burst-Zugriff auf den Speicher
  Cache-Untertstützung
  dynamische Bus-Konfiguration (8, 16, 32-Bit)
  schnelles Busprotokoll
- Hochgeschwindigkeits-XMOS-Technologie
- 84poliges Grid-Array-Gehäuse

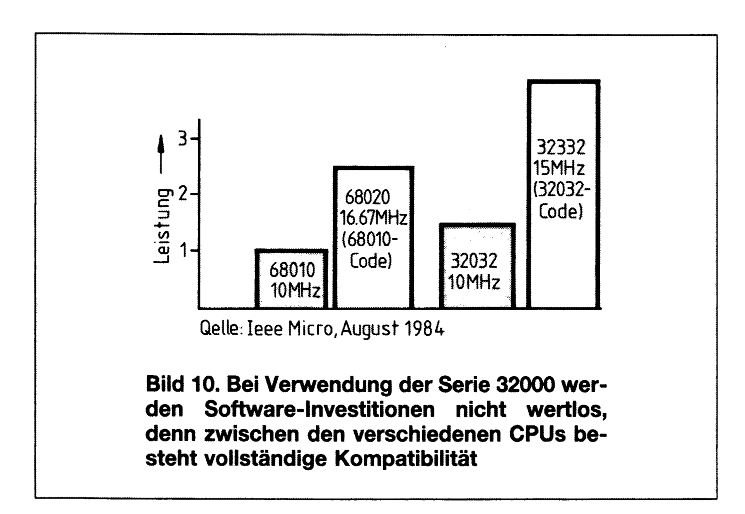


Bild 10. Bei Verwendung der Serie 32000 werden Software-Investitionen nicht wertlos, denn zwischen den verschiedenen CPUs besteht vollständige Kompatibilität

„Es gibt zwei Methoden zum Abschluß von fehlerhaften Buszyklen. Die erste ist die Verwendung eines Software-Handlers zur Emulation des Zyklus und die zweite ist die Möglichkeit, daß der Prozessor die Buszyklen erneut ablaufen lassen kann, nachdem der Fehler beseitigt wurde."

Die Tatsache, daß das Wort „Zyklus" in der Mehrzahl auftritt, ist beunruhigend.

In der CPU 32332 verändert sich in bezug auf die Zustände während der Ausführung eines Befehles nur sehr wenig, so daß der gesamte Zustand beim Beginn eines Befehls auf dem Chip abgespeichert werden kann. Wenn ein Seitenfehler durch einen „Abort" (der sich von einem Busfehler beim NS32332 unterscheidet) angezeigt wird, müssen lediglich die Werte des sichtbaren Registersatzes im Stack gerettet werden. Der PC und die Register MOD/USR werden automatisch von der Hardware des Chips gespeichert; so daß bei dieser CPU acht Byte Information dieselbe Wirkung haben wie bei 64 oder 88 Byte umfassenden Stack-Frames, die vom 68020 gerettet werden.

Es ist naheliegend, daß das Verfahren der Befehls-Weiterführung nicht schneller sein kann als der Befehls-Restart, weil der Teil des Befehls, der bereits ausgeführt ist, nicht noch einmal wiederholt werden muß. Kein Befehl dauert so lange wie das Abspeichern aller Daten im Stack und das anschließende Zurückholen.

Zusammenfassend kann man sagen, daß der Befehls-Restart des 32332 einfach aufgebaut und daher leicht zu verwenden ist. Benutzt wird ein kleiner, einfacher Ausführungs-Status (16 Byte) und die Funktion erfolgt automatisch (keine Software, keine Spezialfälle). Darüber hinaus gibt es zwei separate Leitungen für Bus-Transaction-Fehler und MMU-Abort.

Synchron oder asynchron?

Die Bauelemente der Serie 32000 verfügen über ein synchrones Bus-Interface, während die MC68000-Familie mit einem synchronen Bus arbeitet. Um einen asynchronem Bus an eine beliebige Schaltung anschließen zu können, die über einen internen Zustand verfügt (dazu zählen alle Einheiten, die sich anschließen lassen), muß der Bus mit dem Takt in dem betreffenden Bauelement synchronisiert sein. Dies führt zum Verlust eines Zyklus, was sich nicht vermeiden läßt, insbesondere weil die Phase des Bussteuersignales, das von Bauelementen der 68000-Familie ausgegeben wird, eine Phasen-Unsicherheit von bis zu ½ Takt in bezug auf den Takt der CPU besitzen können. Bei einem synchronen Bus, wie er von der Serie 32000 angeboten wird, treten weder Synchronisation noch Taktverlust auf.

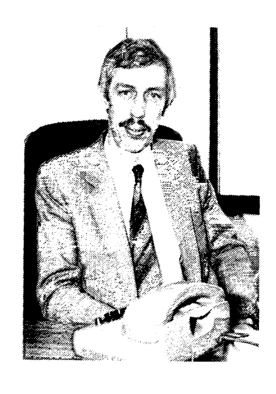
Software-Investitionen bleiben erhalten

Es ist schon häufiger vorgekommen, daß bei traditionellen Mikroprozessorfamilien im Zuge der Weiterentwicklung wichtige Punkte der Architektur und der Programmierumgebung Veränderungen unterzogen wurden. Benutzerprogramme mußten in diesem Fall neu geschrieben werden, um alle Vorteile neuerer Prozessortypen ausnutzen zu können (*Bild 10*). Falls dies nicht gemacht wurde, hat man im Vergleich zu den Problemen einer neuen Mikroprozessorgeneration relativ wenig Vorteile. Bei der Serie 32000 gibt es vollständige Software-Kompatibilität zwischen den CPUs, beginnend vom Typ 32008, über die Bausteine 32016, 32032, 32C016 (die CMOS-Version) bis zum 32332. Dies bedeutet, daß jegliche Software sofort auf einem neuen Prozessor lauffähig ist, ohne daß auch nur eine Zeile neu geschrieben werden muß. Der Schritt zum Übergang auf die nächste Generation ist daher sehr stark erleichtert. Wenn man diesen Punkt in Erwägung zieht, kommt man sehr schnell auf die Bedeutung für die Entwicklungszeit, die Zeit bis zur Marktreife oder die Entwicklungskosten. Software-Kompatibilität ist in diesem Zusammenhang sicherlich eine der wichtigsten Eigenschaften.

Zusammenfassung

Die CPU NS32332 ist im Hinblick auf höchste Systemleistung konzipiert und basiert auf den umfangreichen Erfahrungen von National Semiconductor im Bereich der 32-Bit-Mikroprozessoren. Hiermit bietet sich einem Anwender eine konkurrenzfähige Lösung, die auf der bewährten Architektur beruht und die langfristig einen klaren Weg für Weiterentwicklungen zukünftiger Produkte auf der Basis von 32-Bit-Mikroprozessoren bietet.

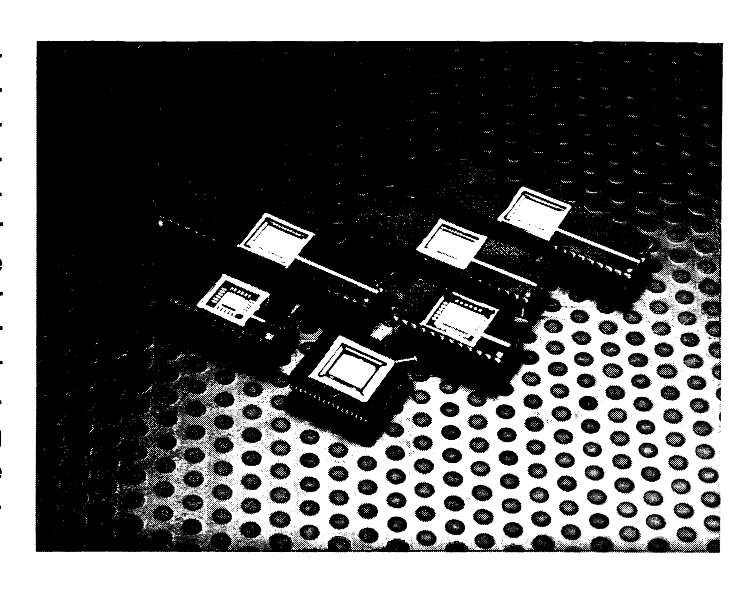
Jean Claude Mathon ist als Verkaufsleiter von Mikroprozessoren und Systemen der französischen Niederlassung von National Semiconductor für den Verkauf, die technische Unterstützung und die Einführung von den Advanced Microprocessors, Systeme und OEM Microprocessor Boards in Frankreich verantwortlich. Jean Claude Mathon, gebürtiger Franzose, ist seit 1983 bei National Semiconductor Corporation beschäftigt.
Vorher war er Laborleiter für 32000 Mikroprozessoren bei einem Halbleiterhersteller und Technischer Manager eines Herstellers von OEM boards.



Jean-Claude Mathon

Effizienz beim Entwickeln mit der 32000-Familie

Mit dem Umsteigen einer wachsenden Zahl von OEM-Anwendern und System-Integratoren auf 32-Bit-Mikroprozessoren stehen die Anwender der 68000- und 68010-Mikroprozessoren vor einer kritischen Entscheidung: Entweder gilt es, die Systeme unter Verwendung der CPU 68020 auszubauen, oder sich der Mehrheit der 32-Bit-Anwender anzuschließen und die Serie 32000 zu verwenden. Der Ausbau innerhalb der 68000-Familie ist möglich und war für viele Anwender bei der Entscheidung für diese CPUs ein wichtiger Gesichtspunkt. In der heutigen Praxis aber ist es einfacher und schneller, eine neue 32000-Entwicklung von Null zu beginnen, als innerhalb der 68000-Familie auszubauen. Außerdem wird das neue System leistungsfähiger und billiger sein.



Natürlich muß eine solche Behauptung bewiesen werden. Als erstes ist zu betonen, daß die Anwender nicht unbedingt derselben 8-16-32-Bit-Sequenz folgen müssen, wie es die Technik tat. Diejenigen, die auf 32-Bit-CPUs umstellen, tun dies aus bestimmten Gründen. Die Wahl des Mikroprozessors sollte auf derselben Basis erfolgen. Die Faktoren, die solche Entscheidungen beeinflussen können, werden unter drei Hauptgesichtspunkten betrachtet:
- Entwicklungszeit
- Preis-/Leistungsverhältnis
- Produkt-Lebensdauer.

In allen Bereichen hat die Serie 32000 klare Vorteile gegenüber der CPU 68020, selbst für diejenigen Anwender, die bereits die Prozessoren 68000 oder 68010 benutzen.

Zeit zur Vermarktung

Der Zwang, neue Produkte so schnell wie möglich auf den Markt zu bringen, war noch nie so stark wie heute. Dies ist ein Faktor, der im Prinzip die Alternative „Ausbauen" begünstigt. Wenn man aber abwägt, was zu einem Ausbau alles gehört, entsteht ein ganz anderes Bild. Die CPU 68020 wurde mit dem Ziel entwickelt, einen Chip mit einer Leistungsfähigkeit auszustatten, die mit der des 32332-Prozessors vergleichbar ist, dabei aber die Kompatibilität mit der relativ „primitiven" 68000-Architektur zu wahren. Um dieses Ziel zu erreichen, waren größere Eingriffe in die Architektur nötig, einschließlich Änderungen am Befehlssatz, an den Adressierungsmodi und den internen Registern.

Damit aus diesen Verbesserungen Vorteile entstehen, muß der 68020-Anwender die Anwendungsprogramme komplett umschreiben und am Betriebssystem wesentliche Änderungen vornehmen. Die Alternative wäre, die alten Programme auf der neuen CPU laufen zu lassen, dabei mehr als 30 Prozent der potentiellen Leistungsfähigkeit zu opfern und so den Vorteil, den der Umbau bringt, zunichte zu machen. Falls der Anwender sich für das Umschreiben der Software entscheidet, benötigt er auch neue Compiler.

Sofern die Anwendungen einen virtuellen Speicher erfordern, was bei den meisten heutigen leistungsfähigen Unix-Systemen der Fall ist, ist das Problem noch größer. Es wird ein beträchtlicher Aufwand für die Hardware-Entwicklung notwendig, weil kein Speicherverwaltungs-Chip verfügbar ist, der die anforderungsgesteuerte Verwaltung des virtuellen Speichers unterstützt. Die Übertragung des Betriebssystems auf die neue Hardware, einschließlich größerer Änderungen im Kern, wird ein teures und langwieriges Unterfangen.

Angesichts der Größe des Problems kann es kaum überraschen, daß manche 68000-Benutzer den Versuch, ihr System auszubauen, aufgegeben und stattdessen auf die Serie 32000 umgestellt haben. Das komplette 32000-Angebot, einschließlich der CPU, Speicherverwaltungs- und Gleitkomma-Einheit, ist in größeren Stückzahlen verfügbar. In dem System gibt es keine Lücken, die mit Anwender-Hardware gefüllt werden müßten. Deshalb kann sich der Benutzer voll auf seine Anwendung konzentrieren.

Der kompletten Hardware-Lösung, die der 32000-Chip-Set bereitstellt, steht eine Auswahl funktionsfähiger Unix-Betriebssysteme zur Seite. Im Gegensatz zur 68020 enthalten die für die 32000 verfügbaren Unix-Versionen die neuesten Releases von „4.1-bsd", „4.2-bsd" und „System-V". Da die Speicherverwaltungs-Hardware auf Silizium implementiert wurde, ist der anforderungsgesteuerte virtuelle Speicher in das Betriebssystem eingebaut, so daß der Benutzer den Kern nicht zu modifizieren braucht.

Anwender, die keine eigene Hardware bauen wollen, oder die den schnellstmöglichen Weg zum marktfähigen Produkt brauchen, können Zielsysteme wie das „ICM3232" einsetzen, ein Mehrzweck-Entwicklungs- und Zielsystem, oder eine Reihe der Höchstleistungs-Add-in-Karten für den IBM-PC. Jedes davon ist eine vollständige Hardware-/Software-Lösung mit volltransparentem, anforderungsgesteuertem virtuellen Speicher und einer Betriebssoftware „System-V". Nationals System-V-Port enthält rekonfigurierbare binäre I/O-Treiber. Das heißt, daß das endgültige Port zum Benutzer-Zielsystem üblicherweise auf die I/O-Treiber beschränkt ist. Infolgedessen braucht der Anwender den Kern nicht zu modifizieren und spart auch die beträchtlichen Ausgaben für die Unix-Source-Lizenz.

Unterstützung auf Hochsprachen-Ebene

Es ist unbestritten, daß die Software-Entwicklungszeit durch die Verwendung von Hochsprachen stark reduziert werden kann. Auf der anderen Seite ist die Leistungsfähigkeit des Systems normalerweise höher, wenn der Source-Code in Assemblersprache geschrieben wird. In der Praxis wird eine Mischung von Hochsprache und Assemblersprache verwendet, wobei das Verhältnis zwischen beiden vom Zeitverhalten, den Leistungsansprüchen und der Effizienz des Compilers abhängt. Je besser der Compiler ist, desto kleiner ist der Anteil an Assembler-Codes.

Die 32000-Architektur wurde von Anfang an dafür konzipiert, effiziente Compiler zu unterstützen. Sie unterstützt direkt Befehle auf Hochsprachen-Ebene wie „CASE" und höhere Datenstrukturen wie Arrays und Records. Am wichtigsten aber ist, daß der Befehlssatz vollständig symmetrisch und orthogonal ist. Das bedeutet, daß jeder Befehl mit jedem relevanten Datentyp und jedem Adressierungsmodus eingesetzt werden kann. Es gibt keine Einschränkungen oder Sonderfälle, die die Compiler-Effizienz beeinträchtigen.

Die CPU 68020 hat im Gegensatz dazu eine geringe Symmetrie. Ihr Befehlssatz ist voller Sonderfälle, jede Kombination von Befehl, Datentyp und Adressierungsmodus muß auf Zulässigkeit geprüft werden. Folglich können für die 68020 geschriebene Compiler nicht an die Effizienz von 32000-Compilern heranreichen. 68020-Anwender sind gezwungen, einen größeren Teil ihres Codes in Assemblersprache zu schreiben oder Einbußen bezüglich der Leistungsfähigkeit hinzunehmen. Was noch gravierender ist: Die architektonischen Einschränkungen, die zu ineffizienten Compilern führen, machen auch das Schreiben von Assembler-Programmen schwieriger.

Preis/Leistungsverhältnis

Zwar hat jede Anwendung ihre eigenen Anforderungen an das Preis/Leistungsverhältnis, die wichtigsten Parameter dafür lassen jedoch keinen Zweifel daran, daß die 32000-Architektur auf diesem Gebiet gewinnt. Hierbei ist der wichtigste Punkt, auf welcher Basis der Leistungsvergleich angestellt wird. Die 68020 wurde als Hochleistungs-CPU entwickelt, während die 32332 als Teil eines Hochleistungssystems entworfen wurde. Die Schöpfer der 32332 waren nicht durch die Forderung eingeschränkt, die Kompatibilität mit einer „altmodischen" Architektur zu bewahren, sondern konnten in die zukunftssichere 32000-Architektur ganz wesentliche System-Eigenschaften implementieren, die die beachtliche Leistungssteigerung ohne Befehlsänderungen ermöglichte.

In der Praxis sind einige der Eigenschaften, die in der 68020-Spezifikation gut aussehen, innerhalb eines realen Systems weniger attraktiv. Der interne Cache z. B. ist ein Cache nur für Befehle und enthält nur 64 Worte mit jeweils 32 Bit. Zu einem Befehl im Cache wird in zwei Taktzyklen zugegriffen im Gegensatz zu den drei oder mehr Zyklen, die für externen Zugriff erforderlich sind. Das heißt aber nicht, daß die Leistung um 33 % erhöht wurde.

Zunächst einmal kann der Cache überhaupt keine Zugriffe auf Operanden abhandeln. Da eine CPU typischerweise ebensoviel Zeit mit dem Zugriff auf Daten wie mit dem Lesen von Befehlen verbringt, ist die maximale Verbesserung sofort auf rund 16 % halbiert. Zwei-

tens verwendet die 68020-CPU 16-Bit-Befehlswörter und diese gelten nur dann als „Cache-Treffer", wenn sie zufällig mit Doppelwortgrenzen zusammenfallen. Statistisch gesehen ist die Chance dieses Ereignisses 50 %, was die maximale Leistungssteigerung nochmals auf 8 % halbiert. Angesichts des großen Overheads durch die Cache-Nichttreffer ist es sehr zweifelhaft, ob der interne Cache des 68020 überhaupt irgendeinen Vorteil bringt.

Es gibt keinen Zweifel, daß bedeutende Leistungsverbesserungen durch einen internen Cache, der Zugriffe sowohl auf Daten als auch auf Befehle behandeln kann, erreicht würden. Bis die Fertigungs-Technik das ermöglicht, ist es besser, einen externen Cache zu verwenden. Weiterentwickelte 32000-CPUs haben Eigenschaften, die externe Caches wirkungsvoll unterstützen.

Die Auswertung der Befehls-Ausführungszeiten der 68020 ist ein weiterer Punkt, an dem Vorsicht angebracht ist. Mit einem Takt von 16 MHz braucht die 68020 zum Zugriff auf einen externen Cache zwei Wartezyklen, so daß zur Basis-Operationszeit für jeden Operanden zwei weitere Taktzeiten addiert werden müssen. Um aus dem 16-MHz-Takt vollen Nutzen zu ziehen, ist ein externer Cache erforderlich. Die Cache-Komparatoren müssen innerhalb von 55 ns reagieren. Solche Bauteile sind gegenwärtig nicht verfügbar. Falls sie auf den Markt kommen, müssen sie in der teuren Bipolar-Technologie hergestellt werden. Die schnellsten verfügbaren Cache-Komparatoren haben eine Reaktionszeit von 117 ns, weit außerhalb der 68020-Anforderungen. Der Zugriff auf den normalen lokalen oder Systemspeicher erzeugt nahezu dieselbe Anzahl von Wartezyklen.

Obwohl die Leistungsfähigkeit der 68020-CPU mit der 32332-CPU vergleichbar ist, solange man die CPUs isoliert betrachtet, kann ein auf 68020 basierendes System bezüglich Leistung an ein 32000-System nicht heranreichen. Genaue Leistungsvergleiche sind nicht leicht anzustellen, weil jeder 68020-Benutzer seine eigene Speicherverwaltungs-Hardware entwickeln und seinen Betriebssystem-Kern umschreiben muß. Das ist eine mühsame Aufgabe, da die 68020 nicht für die Unterstützung von anforderungsgesteuerter Speicherverwaltung entworfen wurde, der einzigen Form von virtuellem Speicher, die für Hochleistungs- und Mehrbenutzer-Systeme akzeptabel ist.

Die 68020 hat z. B. keine echte Möglichkeit des „Befehlsabbruchs mit Wiederholung", die zur effizienten Behandlung von Seiten-Fehlern unabdingbar ist. Das Abort-Signal der 68020 benutzt die Bus-Error-Leitung. Die Antwort auf eine Abort-Situation ist langsam, weil 88 Byte an internem Status gerettet werden müssen. Befehle werden nicht, wie in 32332-Systemen, automatisch wiederholt. Hingegen ist eine von der Software gesteuerte Wiederholung erforderlich.

Der vielleicht wichtigste Leistungsparameter ist die Compiler-Effizienz, da fast alle für 68020- oder 32332-Systeme geschriebene Software aus Hochsprachen generiert wird. Wie oben beschrieben, sind 32000-Compiler weitaus effizienter als die für irgendeine andere CPU geschriebenen, weil die Architektur mit der Compiler-Effizienz als Hauptziel entwickelt wurde. Nur in 32000-Systemen ist compilierter Code fast so kompakt wie assemblierter.

Die Verbindung einer auf Hochsprachen optimierten Architektur mit der eingebauten Unterstützung des anforderungsgesteuerten virtuellen Speichers macht die 32000-CPUs zu den leistungsfähigsten Unix-Maschinen der Industrie. In Ermangelung dieser Eigenschaften kann die 68020 kaum als eine Unix-Maschine betrachtet werden. Laut unabhängigen Tests läuft selbst Intels 80286 mit 12,5 MHz Taktfrequenz der CPU 68020 mit 16 MHz in einer Unix-Umgebung den Rang ab. Aber kaum jemand, außer Intel, würde behaupten, daß die 80286 eine bessere Unix-Maschine sei als die CPU 32332.

Ein führender europäischer Zweitlieferant für Intel-Mikroprozessoren benutzt z. B. den Baustein 32332 für seine neuen Unix-Supermikros. Der IBM-Personal-Computer ist auf einem Intel-Mikroprozessor aufgebaut, aber fast alle unabhängigen Anbieter von Zusatzkarten für den IBM-PC verwenden die 32032.

Trotz ihrer überlegenen Leistungsfähigkeit kosten 32000-Systeme weniger als 68020-Systeme. Genaue Preisvergleiche sind schwer anzustellen, weil die 68020-CPU Teil eines noch nicht vollständigen Chipsets ist. Was aber feststeht ist, daß der 68020-Chipset immer teurer sein wird als ein 32000-Chipset. Zum Beispiel kostet bei einer Stückzahl von 1000 oder mehr pro Jahr ein vollständiger 32000er-Satz für 10 MHz (32332-CPU, -MMU, -FPU, -TCU, -ICU) weniger als 350 $, was geringfügig über dem Preis der nackten 68020 liegt. Wenn man statt der 32332-CPU die 32016-CPU einsetzt, die nur in der Breite des externen Datenbusses abweicht, beträgt der Preis für dasselbe komplette Paket bei 10 MHz nur 180 $ und bei 6 MHz 60 $.

Die FPU 68881 kostet etwa zehnmal mehr als die FPU 32081. Die 68881 führt zwar mehr Operationen aus als die 32081, aber diese zusätzlichen Operationen werden vielleicht von weniger als 10 % der Benutzer benötigt. Die übrigen 90 % der Benutzer müssen einen hohen Preis für Funktionen zahlen, die sie nicht brauchen.

Produkt-Lebensdauer

Nichts vom bislang dargestellten Sachverhalt tut einer wirklichen Ausbau-Strategie Abbruch. Es ist wichtig sicherzustellen, daß Produkte schnell verbessert werden können, um aus technischen Entwicklungen Nutzen zu ziehen. Das Problem der 68000-Anwender ist es, daß der Weg „Ausbau" näherliegender als realistisch ist. Aber,

den Beharrlichen, denen es gelingt, die Schwierigkeiten zu überwinden, bleibt ein letztes Problem: was können sie für ihre nächste Ausbaustufe tun?

Motorola hat bisher keine Pläne über weitere Verbesserungen der 68000-Architektur veröffentlicht. Es ist schwer zu erkennen, welche wesentlichen Verbesserungen an einer Architektur angebracht werden könnten, so daß man den Eindruck gewinnt, daß sie schon an ihrem Ende angelangt ist. Die Möglichkeit des 68000-Ausbaus endet, so wie es jetzt steht, bei der 68020. Zur späteren Leistungssteigerung werden die 68020-Anwender daher gezwungen sein, auf eine neue Architektur umzustellen.

Im Gegensatz dazu sind die 32000-CPUs erst der Anfang einer Aufwärtsentwicklung, die fortgesetzt wird. Die nächste CPU innerhalb der 32000-Familie, die 32C532, deren Einführung Mitte 1987 stattfindet, hat eine dreimal höhere Leistung als die 32332. Sie enthält kleine Änderungen in der internen Schaltung, aber die Architektur wurde beibehalten. Alle existierenden 32000-Programme laufen darauf.

Zusammenfassend muß betont werden, daß die Techniken, die zum Vergleich früherer Mikroprozessoren verwendet wurden, zum Vergleich von VLSI-Einheiten nicht geeignet sind. Heute ist der wesentliche Gesichtspunkt die Architektur. Die 32000-Architektur liefert Leistung sowohl auf System- wie auf Chip-Ebene und bietet einen einzigartigen Weg für den Ausbau. Dies, und nicht die Taktfrequenz, ist der Maßstab, an dem neue 32-Bit-Mikroprozessoren gemessen werden müssen.

Die IBM-PC-Cross-Entwicklungs-Software

Da zahlreiche DV-Systementwickler heute den IBM-PC für vielfältige Zwecke einsetzen, bietet National Semiconductor eine Cross-Entwicklungs-Software zur Evaluierung von 32000-Software auf dem IBM-PC an. Das Cross-Software-Paket ist Bestandteil des Toolkit, welches zum Standard-Lieferumfang des NS32032-Starter-Kit gehört und als Option auch zum NS32016-Starter-Kit lieferbar ist.

Die wesentlichen Elemente der Cross-Entwicklungs-Software von OWL Computer Inc. mit der Bezeichnung NSX-32 sind:

- *ASM32 – Ein Macro-Assembler für die Serie 32000, der einen verschiebbaren Code erzeugt, komplexe Ausdrücke akzeptiert und blockstrukturierte Programme unterstützt.
- *LINK32 – Ein Linker zum Verbinden von ASM32- oder HLL-compilierten Files zu verarbeitbaren Einheiten.
- *LIB32 – Zur Erzeugung, Pflege, Adressierung und zum Updating von NSX-32-Files auf Modul-Ebene, z. B., um Standard-Module zu generieren.
- *BIN32 – Zur Konvertierung ablauffähiger Files in ein Format, das für PROM-Programmierer mit 8-, 16- oder 32-Bit-Wortbreite geeignet ist.
- *DEBUG32 – Ein interaktiver, symbolischer Debugger für Anwenderprogramme in Assembler, „C“ oder Pascal; inklusive Disassembler, direkten Zugriff auf den Speicher und leistungsfähiger Unterbrechungsmöglichkeiten.
- *MON32 Quellenprogramm des Monitors MON32 mit allgemeinen Monitorfunktionen und einem Programm für die Kommunikation mit DEBUG32.
- *TESTFILES – TEST/DEMO-Assembler-Quellenprogramm-Files, z. B. arithmetische Ausdrücke, Assembler-Test, Befehlssatz-Test, Macro-Testprogramm, Fließkomma-Test.

Als Option:

- *iDEBUG32 – Ein erweiterter Debugger für Starter-Kit-Anwender, die mit einen Realzeit-In-System-Emulator arbeiten wollen, um Hardware und Software auszutesten.

Wird auf höchste Optimierung der Sopftware Wert gelegt, sollten die optimierten Compiler und Debugger von National Semiconductor auf professionellen Entwicklungssystemen, wie SYS32, VR32 oder VAX unter VMS oder UNIX eingesetzt werden. Geeignet ist dafür auch ein IBM-PC mit 32000-Erweiterungskarte, die von National unter der Bezeichnung SYS32/20 angeboten wird und mit der VAXähnliche Leistung auf einem PC erreicht wird. Mit dieser 32000-Erweiterungskarte sind im IBM-PC alle Compiler unter System V Unix von National ablauffähig.

Dr. Werner Trattnig

NS32000-Familie:

Maßgeschneidert für höhere Sprachen

Seit Jahren versuchen Projektmanager verzweifelt, die Produktivität bei der Entwicklung von Software zu steigern. Höhere Sprachen sind ein Mittel dazu. Doch allzu oft programmiert man lieber in Assembler, weil der Zielprozessor Hochsprachen nur unzulänglich unterstützt und der Compiler entsprechend ineffektiven Code liefert. Die NS32000-Familie von National Semiconductor räumt diesen Mangel aus. – Aufgrund welcher Eigenschaften sie dazu in der Lage ist, erfahren Sie auf den nächsten Seiten.

Die Entwickler von Mikroprozessoren haben die Vorteile der Orthogonalität und Symmetrie (siehe Kasten) seit vielen Jahren erkannt, aber sie konnten sie nicht voll realisieren, weil sie im Widerspruch zu anderen Forderungen standen. Die meisten Mikroprozessoren haben z. B. eine feste Befehlswort-Länge. Im Falle des 68000 wird ein 16-Bit-Befehlswort verwendet, das 65 536 Kombinationen erlaubt. Der flexibelste 68000-Befehl, MOVE, braucht 12 288 Kombinationen, um jede Kombination von Operandenlänge und Adressierungsart abzudecken. Ähnlich flexible Speicher-zu-Speicher-Versionen von ADD, AND, EOR usw. würden jede für sich weitere 12 288 Operationscode-Kombinationen erfordern. Deshalb mußte die Symmetrie dieser Befehle reduziert werden, damit die feste Befehlslänge beibehalten werden konnte.

Im Gegensatz dazu waren Orthogonalität und Symmetrie primäre Ziele bei der Entwicklung der Architektur der 32000-Familie, und andere Entwicklungsziele wurden diesem Ziel untergeordnet. Anstelle einer festen Befehlslänge variieren die Codes in ihrer Länge von einem Byte bis drei Byte. Die Befehlscodes wurden so zugewiesen, daß die am häufigsten verwendeten Befehle kurze Codes haben, während die längeren Operationscodes für weniger oft verwendete Befehle benutzt werden.

Diese Eigenschaften gestatten dem Programmierer, Assembler-Programme effizienter zu schreiben. Sie führen auch zu effizienterer Ausführung von Assembler-Code, aber sie verringern nicht die „semantische Lücke" zwischen Assembler- und Hochsprachen-Ebene, die ebenfalls zur Compiler-Effizienz beiträgt. Es gibt in Hochsprachen vier wichtige Konzepte – Befehle zur Steuerung des Programmablaufs, Prozeduren, komplexe Datenstrukturen und Module –, die von bisherigen Mikroprozessor-Architekturen nicht unterstützt wurden.

1 Steuerung des Programmablaufs

Die 32000-Architektur bietet direkte Unterstützung für die CASE- und FOR-Befehle, die man in den meisten Hochsprachen findet. Die meisten Mikroprozessoren

Die 32-Bit-Prozessorfamilie NS32000 wurde ursprünglich von National Semiconductor konzipiert und wird inzwischen von Texas Instruments als Zweitlieferant angeboten (Bild: National Semiconductor)



erfordern, daß Schleifen als drei separate Befehle ausgeführt werden. Zuerst wird ADD verwendet, um eine feste Schrittweite auf einen Schleifen-Index zu addieren. Als nächstes prüft ein COMPARE-Befehl den neuen Index auf eine obere Grenze. Als letztes wird ein Branch-Befehl eingesetzt, um an den Startpunkt der Schleife zurückzukehren, falls die Obergrenze nicht erreicht wurde. Der ACB-Befehl (ADD, COMPARE and BRANCH) des 32000 realisiert das Schleifen-Konstrukt in einem einzigen Befehl.

Der CASE-Befehl, wie er in PASCAL und anderen Hochsprachen verwendet wird, bietet eine Verzweigung, die das Programm an einer Stelle fortsetzt, die mit Hilfe des Wertes einer Variablen ermittelt wird. Da nur wenige Mikroprozessoren die Indizierung in Verbindung mit BRANCH-Befehlen erlauben, erfordern Verzweigungen üblicherweise mehrere Maschinenbefehle. Der CASE-Befehl des 32000 erlaubt die direkte Implementierung einer Verzweigung in einem einzigen Befehl, und zwar mit Hilfe der skalierten Indizierung und einer Tabelle von Sprungadressen, die irgendwo im Speicher stehen kann und über ein Allzweckregister adressiert wird.

2 Prozeduren

Obwohl die Zahl der heute gebräuchlichen Hochsprachen groß ist, basieren die meisten der populären modernen Sprachen auf dem Prozedur-Konzept. Prozeduren sind die Bausteine, aus denen Anwendungsprogramme gebaut werden. Jede Prozedur führt eine spezielle Funktion aus und kommuniziert mit anderen Prozeduren mit Hilfe von definierten Schnittstellen.

Eine Hochsprachen-Prozedur wird dadurch aufgerufen, daß ein Assembler-Unterprogramm angesprungen wird. Fast jeder Mikroprozessor verfügt über die Basis-Befehle für Unterprogramme: CALL und RETURN. Wichtiger dagegen ist die Art und Weise, wie Argumente und Ergebnisse zwischen Prozeduren weitergeleitet werden. Traditionell verwenden Assembler-Programmierer zwei Techniken: die Parameterübergabe im Speicher oder im Stack.

Zur Unterstützung prozedurorientierter Hochsprachen muß der Mikroprozessor effizientere Mechanismen zum Prozeduraufruf und zur Parameterübergabe zwischen Prozeduren haben. Die 32000-Familie löst dieses Problem, indem sie zwei Spezialregister, die statische Basis (SB) und den Frame-Pointer (FP) sowie drei Adressierungsarten – „Speicherbereich", „Speicher relativ" und „Top of Stack" (TOS) – zur Verfügung stellt. Die Funktion der Register ist in *Bild 1* dargestellt, und zwar unter Verwendung einer hypothetischen Prozedur, die das Signal eines Thermofühlers linearisiert.

Weil ein Thermofühler ein nichtlineares Element ist, wird er normalerweise kalibriert. Die Kalibrierkurve wird dann verwendet, um die wirkliche Temperatur zu bekommen, die der gemessenen Ausgangsspannung entspricht. Ein Weg, wie man dies tun kann, ist, sich der Kalibrierkurve schrittweise durch lineare Segmente anzunähern, die Koordinaten der Stützpunkte in einer Tabelle im Speicher abzulegen und dann einen geeigneten Interpolations-Algorithmus zu verwenden, um die Temperatur zu berechnen.

Eine Prozedur TEMP(V), die die der Spannung V entsprechende Temperatur berechnet, würde zwei Arten von Daten verwenden. Die Tabelleneinträge, die die Kalibrierkurve des Gerätes beschreiben, sind die globalen oder statischen Daten. Der Parameter V dagegen, ebenso wie alle temporären in den Berechnungen verwendeten Variablen, ist eine lokale oder dynamische Variable. Lokale Variable werden bei jedem einzelnen Aufruf der Prozedur zugewiesen. Sofern mehrere Tasks

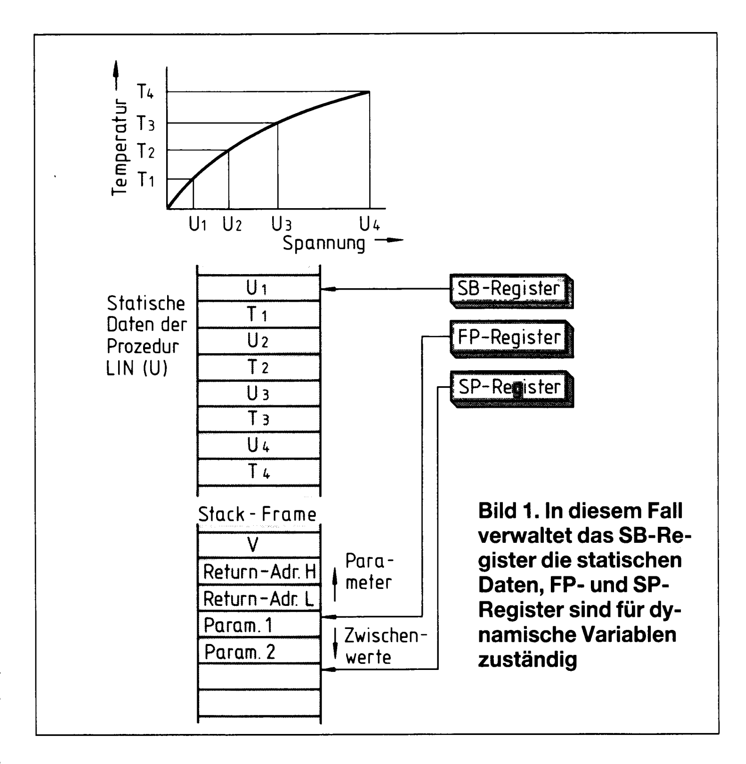


Bild 1. In diesem Fall verwaltet das SB-Register die statischen Daten, FP- und SP-Register sind für dynamische Variablen zuständig

die Prozedur gleichzeitig aufrufen, wird jedem einzelnen Aufruf sein eigener kleiner RAM-Bereich zugeordnet. Er heißt Aktivierungssatz oder Frame, in dem lokale Variable abgelegt werden.

Normalerweise sind die SB- und FP-Register dazu da, auf die statischen bzw. dynamischen Datenbereiche zu verweisen, und die Speicherbereich- und Speicher-relativ-Adressierungsart dienen dem Zugriff auf die einzelnen Datenelemente. In der Speicherbereich-Adressierungsart wird eines der vier speziellen Register (PC, SP, FP, SB) als Basis-Pointer herangezogen. Zum Inhalt des Basis-Pointers wird ein Abstand addiert. Daraus ergibt sich die Operanden-Adresse.

Die Speicher-relativ-Adressierung wird verwendet, wenn der Parameter im Stack eine Adresse oder ein strukturierter Datentyp wie ein Satz oder ein String ist. *Bild 2* zeigt, wie die Operandenadresse in diesem Modus, in dem zwei Abstände angegeben sind, errechnet wird. Der erste Abstand wird zum Basis-Pointer (SP, FP oder SB) addiert. Dies ergibt die Adresse eines zwei-

ten Pointers. Der zweite Abstand wird zu der Adresse addiert, die im ersten Pointer steht. Dies ergibt die endgültige Operanden-Adresse. Dieser Modus ist besonders nützlich, wenn auf Sätze zugegriffen wird, da der zweite Abstand ein Feld innerhalb des Satzes bezeichnen kann.

Die Speicherbereich-Adressierung wird benutzt, um auf Prozedur-Parameter und temporäre Variable unter Verwendung des FP-Registers als Basis-Pointer zuzugreifen. Wie in Bild 2 dargestellt, wird auf Prozedur-Parameter durch Addition eines positiven Abstandswertes zum Inhalt von FP zugegriffen, während temporäre Variable mit negativen Abstandswerten erreicht werden.

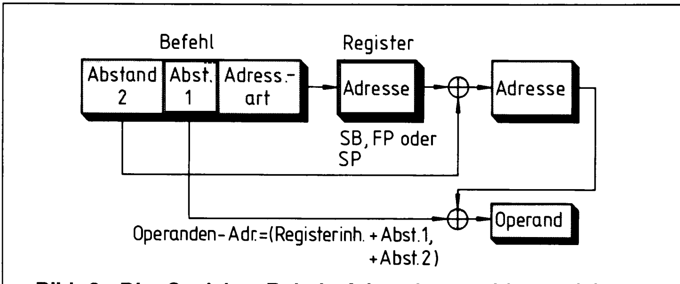


Bild 2. Die Speicher-Relativ-Adressierung bietet sich an, wenn der Parameter im Stack eine Adresse oder ein strukturierter Datentyp ist

Da jeder Prozeduraufruf erfordert, daß ein neuer Stack-Frame aufgebaut wird, steht ein spezieller Befehl zur Verfügung, der dies sehr effizient tut. Der ENTER-Befehl generiert einen Stack-Frame wie folgt:

- Der augenblickliche Inhalt des FP wird in den Stack gebracht.
- Der augenblickliche Inhalt des SP wird in den FP kopiert.
- Ein im Befehlscode angegebener Wert wird vom SP subtrahiert, wodurch der Stack um diese Anzahl von Bytes erweitert wird. Dies hat den Effekt, daß ein Stackbereich als lokaler Speicher reserviert wird.
- Einzelne oder alle Allzweckregister, wie in einem 8-Bit-Feld im Befehlscode angegeben, werden in den reservierten Stackbereich gerettet.

Der ENTER-Befehl steht normalerweise am Anfang des Codes der Prozedur. Ein entgegengesetzter Befehl, EXIT, kann am Ende der Prozedur verwendet werden, um den alten FP und die Allzweckregister, die durch ENTER gerettet wurden, wiederherzustellen und dadurch den alten Frame-Bereich freizugeben und den vorhergehenden Frame wiederherzustellen. ENTER und EXIT verringern den üblichen Unterprogramm-Overhead dadurch, daß sie die Rettung und Wiederherstellung jeder beliebigen Kombination von Allzweckregistern beim Beginn und am Ende des Unterprogramms mit einem einzigen Befehl zulassen, und zwar ohne die unnötigen Verzögerungen, die eintreten, wenn alle Register automatisch gerettet werden.

Array-Element	Adresse
A [1,1,1]	1000
A [1,1,2]	1004
A [1,2,1]	1008
A [1,2,2]	100C
A [1,3,1]	1010
A [1,3,2]	1014
A [2,1,1]	1018
A [2,1,2]	101C
A [2,2,1]	1020
A [2,2,2]	1024
A [2,3,1]	1028
A [2,3,2]	102C

Bild 3. Array-Definition in Pascal und Lage der Elemente im Speicher. Die Basisadresse ist 1000

3 Datenstrukturen

Die von allen Mikroprozessoren unterstützten primitiven Datentypen, d. h. diejenigen Datentypen, die direkt durch Operatoren aus dem Befehlssatz bearbeitet werden können, sind relativ einfache Typen, z. B. ganze Zahlen, BCD-Ziffern, Bitfelder und Gleitkommazahlen. Für Hochsprachen dagegen ist es charakteristisch, daß sie häufig strukturierte Datentypen verwenden. Die drei gebräuchlichsten Strukturen sind Arrays, Strings und Sätze.

Bei den meisten Mikroprozessor-Architekturen erfordert das Auffinden des Speicherplatzes, an dem ein bestimmtes Array-Element gespeichert ist, beträchtlichen Rechenaufwand. Ein Array ist im Speicher immer als eine lückenlose Tabelle abgelegt. *Bild 3* zeigt das Speicherschema, das in C, Pascal und den meisten gebräuchlichen Hochsprachen verwendet wird. Gegeben sei ein Element A(I,J,K...,Z) in einem mehrdimensionalen Array. Man kann die relative Lage des Elementes innerhalb der Tabelle durch Auswertung des Ausdrucks

(...((I*DJ+J)*DK+K)*...)*DZ+Z

berechnen, wobei DI, DJ, DK...DZ die Längen in den Dimensionen I, J, K...Z sind, und zwar auf Null normiert durch Subtraktion ihrer unteren Grenzwerte.

Die Betrachtung der Formel offenbart, daß sie nicht ganz so bequem ist, wie sie zunächst aussieht. Sie kann durch sukzessive Ausführung der Operation

Ergebnis = (vorangehendes Ergebnis) * Dimension + Index

ausgewertet werden, wobei die Parameter „Dimension" und „Index" zuerst DJ und J, dann DK und K sind, bis hin zu DZ und Z. In CPUs der Serie 32000 wird diese Operation durch einen einzigen INDEX-Befehl ausge-

führt, was den Zugriff auf jedes beliebige Element eines N-dimensionalen Arrays nach N-1 Wiederholungen des INDEX-Befehls ermöglicht.

Der eindimensionale Index innerhalb des Arrays, der mit dem INDEX-Befehl errechnet wird, stimmt nur bei Arrays aus 1-Byte-Elementen. Wenn jedes Array-Element N Byte lang ist, muß der Index mit N multipliziert werden. In der 32000-Architektur ist es meistens nicht erforderlich, diese Multiplikation explizit durchzuführen. Statt dessen wird die skalierte Index-Adressierung angewandt. In diesem Modus wird eines der Allzweckregister als Indexregister eingesetzt, sein Inhalt mit 1, 2, 4 oder 8 multipliziert und dann zu einer Basisadresse addiert, die in irgendeiner anderen Adressierungsart angegeben ist. Das Ergebnis ist die endgültige Operanden-Adresse.

Die Speicher-Relativ-Adressierung ist auch besonders nützlich bei der Adressierung von Sätzen, obwohl die Art, wie sie verwendet wird, davon abhängt, wie die Sätze organisiert sind. Eine übliche Technik ist, die Anfangsadresse jedes Satzes in einer Pointer-Tabelle zu halten. Durch Verwendung der Speicher-Relativ-Adressierung kann der erste Abstandswert den Satz angeben, während der zweite Abstandswert auf ein Feld innerhalb des Satzes weist.

Die 32000-Architektur unterstützt Strings durch spezielle Befehle, die die am häufigsten gebrauchten String-Operationen direkt implementieren: Die Übertragung eines Strings von einem Speicherbereich in einen anderen, den Vergleich zweier Strings und das Suchen in einem String nach einem bestimmten Zeichen. Einige frühere Mikroprozessoren enthielten zwar eine Art String-Unterstützung in Form von Blockübertragung oder ähnlichen Befehlen, die 32000-Stringbefehle sind aber wesentlich leistungsfähiger.

Wegen der Komplexität der String-Befehle und der Notwendigkeit, sie unterbrechbar zu machen, werden nicht die allgemeinen Adressierungsarten verwendet. Statt dessen werden spezielle Allzweckregister zum Verweis auf die Strings verwendet, die irgendwo im Speicher stehen können. Vor der Anwendung eines der drei Befehle MOVS (Move String), CMPS (Compare String) oder SKPS (Skip String), werden die Allzweckregister wie folgt geladen:

R0 enthält die Maximalzahl der zu verarbeitenden String-Elemente,
R1 weist auf das erste Element in String 1,
R2 weist auf das erste Element in String 2 (nicht bei SKPS),
R3 weist auf den Anfang einer Übersetzungstabelle, falls erforderlich,
R4 enthält einen Suchbegriff, falls erforderlich.

Die Befehle bearbeiten die String-Elemente der Reihe nach, bis eine angegebene Endebedingung eintritt. Nach der Bearbeitung jedes Elementes wird R0 vermindert, so daß es die Anzahl der noch zu verarbeitenden Elemente enthält. Dabei werden R1 und R2 so modifiziert, daß sie auf die nächsten zu verarbeitenden String-Elemente weisen. Wenn der Wert von R0 auf Null geht, ist der Befehl beendet. Der SKPS-Befehl wirkt auf nur einen String und verändert R2 nicht.

Die Stärke der String-Befehle wird aber noch wesentlich erhöht durch die vier Optionen, die im Befehlscode angegeben werden können. Die Option „Übersetzung"

Der optimale Prozessor – Wunsch und Wirklichkeit

Die Effizienz der Hochsprache ist unabänderlich begrenzt durch die Effizienz der Assembler-Sprache. Wenn es die Architektur dem Assembler-Programmierer nicht gestattet, guten Code zu schreiben, dann wird sie auch einem Compiler nicht gestatten, guten Code zu generieren.

Die Optimierung einer Architektur für Hochsprachen ist daher ein zweistufiger Prozeß. Zuerst muß sie für die Assembler-Ebene optimiert werden, dann müssen weitere Eigenschaften eingeführt werden, um die „semantische Lücke" zwischen Assembler- und Hochsprachen-Ebene zu schließen. Keine Architektur kann für die Assembler-Sprache optimal sein, wenn sie nicht die folgenden drei Hauptbedingungen erfüllt:

Orthogonalität

Die Architektur sollte orthogonal sein, d. h., es sollte einen vollständigen Satz von Operationen für jeden unterstützten Datentyp geben. Wenn z. B. die Datentypen Byte, Wort und Doppelwort (32 Bit) zugelassen sind, dann sollte der vollständige Vorrat an Additions-, Subtraktions-, Multiplikations-, Divisions-, Shift- und logischen Operationen für jede der Integer-Längen zur Verfügung stehen. Der Programmierer sollte sich nicht darum kümmern müssen, ob eine bestimmte Operation für einen gegebenen Datentyp gültig ist. Außerdem sollte die Orthogonalität nicht durch Beschränkung der zur Verfügung stehenden Datentypen erreicht werden.

Die 32000-Familie unterstützt 8-, 16- und 32-Bit-Integer, BCD-Zahlen, Gleitkommazahlen mit einfacher oder doppelter Genauigkeit, Zeichenfolgen, Bits und Bitfelder. Mit den gegenwärtigen Fabrikationsverfahren kann der Gleitkommaprozessor nicht in die CPU integriert werden. Deshalb ist er als getrennter Slave-Prozessor realisiert. Die Gleitkommabefehle sind dagegen im Befehlssatz der CPU enthalten.

Symmetrie

Der Befehlssatz sollte symmetrisch sein, d. h., jeder Operator (Addition, Subtraktion, MOVE usw.) sollte in der Lage sein, zum Zugriff auf die Operanden jede Adressierungsart zu verwenden. Die meisten Befehlssätze von Mikroprozessoren zeigen sehr wenig Symmetrie, so daß jede Kombination von Operator und Adressierungsart auf Gültigkeit überprüft werden muß. Eine häufige Einschränkung ist, für die Spezifikation des Quell-Operanden eines Zwei-Operanden-Befehls eine allgemeine Adressierungsart zuzulassen, für den Ziel-Ope-

bewirkt, daß jedes Element von String 1 vor der Übertragung oder Prüfung übersetzt wird. Dies wird dadurch bewerkstelligt, daß der ursprüngliche Wert des Elementes als Index in einer Übersetzungstabelle verwendet wird, deren Anfangsadresse in R3 steht. Obwohl im allgemeinen die Elemente eines Strings Bytes, Worte oder Doppelworte sein dürfen, steht die Übersetzung nur für Byte-Strings zur Verfügung. Diese Einschränkung ist von keiner praktischen Konsequenz, weil, selbst wenn die Übersetzung zugelassen wäre, die Länge der Übersetzungstabelle für längere Elemente viel zu groß würde.

Die Option „Rückwärts" veranlaßt eine String-Operation in entgegengesetzter Richtung, indem sie Elemente mit fallenden Speicheradressen fortschreitend bearbeitet. Dies wird benutzt, um ein Überschreiben zu vermeiden, was bei anderen MOVS-Befehlen passieren kann, wenn die Strings im Speicher überlappen.

Die Option „Bis Übereinstimmung" bringt den Befehl dazu, abzubrechen, wenn das Element aus String 1 (nach der Übersetzung, falls verlangt) mit dem Inhalt von R4 übereinstimmt. Es gibt auch die komplementäre Option „Solange Übereinstimmung", die den Befehl beendet, sobald das Element aus String 1 nicht mehr mit R4 übereinstimmt.

Weil die Ausführungszeit eines String-Befehls von der Zahl der zu bearbeitenden Elemente abhängt, sind besondere Vorkehrungen für die Behandlung von Interrupts getroffen, die während der Ausführung eines String-Befehls eintreten. In einem solchen Fall beendet die CPU zunächst die Bearbeitung des laufenden Elements und verändert die Register R0...R2 wie gewöhnlich. Sie rettet dann die Adresse des String-Befehls als Rückkehradresse anstelle der Adresse des nachfolgenden Befehls, bevor auf die Interrupt-Service-Routine verzweigt wird.

Wenn die Interrupt-Service-Routine beendet ist, wird der String-Befehl wieder gestartet, aber, weil R0...R2 auf dem letzten Stand sind, ist der Effekt der, daß die Bearbeitung der Strings an dem Punkt fortfährt, wo der Befehl unterbrochen wurde, sofern die Interrupt-Routine dem normalen Brauch folgt, vor der Rückkehr alle benutzten Register wiederherzustellen.

4 Modulare Programmierung

Fast alle Hochsprachen enthalten Vorkehrungen zur Unterstützung der modularen Programmierung. Pascal z. B. bietet Import- und Export-Anweisungen, mit denen Programm-Module auf Variable, Funktionen oder Prozeduren zugreifen können, die in einem anderen Modul definiert sind. Zumindest theoretisch können Module in einer Bibliothek gesammelt und von mehreren Programmen benutzt oder sogar im ROM fixiert und als Firmware-Baustein behandelt werden.

In der Praxis gibt es aber Schwierigkeiten, die von der unzureichenden Unterstützung der modularen Programmierung herrühren. Software-Module, die compiliert und zusammengebaut wurden, nennt man Objekt-Module und speichert sie üblicherweise als relokativen Objekt-Code. Zur Kombination mehrerer Objekt-Module zu einem Programm fügt ein Linking Loader die Objekt-Module in einem einzigen linearen Adreßraum zusam-

---

randen aber ein Spezialregister, z. B. einen Akkumulator, zu verlangen.

Die 32000-Familie ist in dieser Hinsicht nicht ganz vollkommen, aber die Einschränkungen sind von geringer Zahl und in ihrer Bedeutung vernachlässigbar. Der skalierte Indizierungs-Modus z. B. verlangt, daß der Index in einem der acht 32-Bit-Allzweckregister steht, obwohl die Basisadresse durch jede Adressierungsart spezifiziert werden kann. In fast allen Fällen aber kann jeder Befehl mit jedem relevanten Datentyp und jeder Kombination von Adressierungsarten verwendet werden.

Insbesondere ist der NS32000 eine echte Zwei-Adreß-Maschine. Das bedeutet, daß beide Operanden in einem Befehl wie A:=A+B irgendwo im Speicher stehen können. Der Programmierer braucht keine Zwischenbefehle zu verwenden, um vor der Ausführung der Operation einen der Operanden in ein Register zu bringen. Gegenwärtig hat nur die VAX-Architektur eine vergleichbare Symmetrie.

Allzweckregister

Die Architektur sollte eine angemessene Anzahl von Allzweckregistern bieten. Dies ist ein etwas kontroverser Punkt. Einige Theoretiker haben sogar Architekturen vorgeschlagen, die überhaupt keine Allzweckregister haben. Empirische Untersuchungen haben aber gezeigt, daß die Programme die meiste Zeit damit verbringen, eine kleine Anzahl von Variablen zu bearbeiten. Wenn man diese Variablen in Registern auf dem Chip hält, reduziert dies die Anzahl der externen Speicherzugriffe, was wiederum zu einer schnelleren Programmausführung führt.

Da fast alle Mikroprozessoren Gebrauch von internen Registern machen, ist eine praktischere Überlegung die Anzahl und Art dieser Register. Ganz allgemein kann man Register in zwei Gruppen einteilen: solche, die einem bestimmten Zweck dienen, z. B. als Befehlszähler oder Stackpointer, und solche, die dies nicht tun. Die ersten Mikroprozessoren hatten eine Vielzahl spezieller Indexregister, Akkumulatoren, Datenpointer usw., die nicht untereinander vertauschbar waren. Heute aber geht man im allgemeinen davon aus, daß solche Register, die keinen speziellen Zweck haben, wirklich Allzweckregister sein sollten. Es sollte deshalb möglich sein, ein Allzweckregister als Akkumulator, Datenpointer oder Indexregister zu verwenden. Dies wiederum impliziert, daß die optimale Registerlänge 32 Bit beträgt, da dies die wirkungsvollste Pointerlänge ist.

Untersuchungen haben gezeigt, daß fünf Allzweckregister für die meisten Anwendungen ausreichen. Die 32000-Familie bietet acht, damit das 3-Bit-Feld im Operationscode voll ausgenutzt ist.

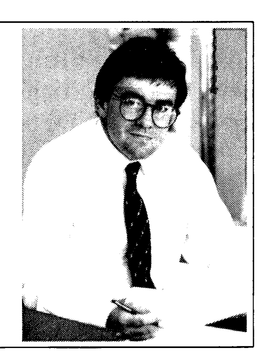
men und setzt dabei für alle nicht aufgelösten Adressen in den Modulen absolute Adressen ein. Die Anpassung der Adressen von Unterprogramm-Aufrufen, Verzweigungen und Datenzugriffen innerhalb eines Moduls nennt man Relokation. Das Besorgen der absoluten Adressen von Unterprogrammen und Variablen, die in anderen Modulen deklariert wurden, nennt man Linken. Die Aufgabe der Relokation und des Linkens von plattenresidenten Modulen vor dem Lauf des Programmes kann eine schwierige und langwierige Übung sein. Da ROM-Adressen nicht geändert werden können, sind ROM-residente Module in den meisten Mikroprozessoren nur unter größten Schwierigkeiten verwendbar.

Die 32000-Architektur bietet sowohl allgemeine als auch spezielle Unterstützung für die modulare Programmierung. Die generelle Unterstützung kommt von den leistungsfähigen relativen Adressierungsarten und der Symmetrie des Befehlssatzes. Alle Datenzugriffe und alle Unterprogramm-Aufrufe, Sprünge und Verzweigungen innerhalb eines Moduls können als Abstand vom Befehlszähler, vom statischen Basis-Register oder vom Frame-Pointer angegeben werden, wodurch wirklich ein lageunabhängiger Code entsteht.

Lageunabhängiger Code zusammen mit dem virtuellen Speichermechanismus der 32000-Familie, der Seiten nach Bedarf verwaltet, bedeutet, daß ein Programm überall im virtuellen oder physikalischen Adreßraum liegen kann und dennoch korrekt abläuft. Unabhängig von der Länge des Programms sind niemals Änderungen in Sprungbefehlen notwendig, weil der vorzeichenbehaftete 30-Bit-Abstandswert über den gesamten logischen Adreßbereich hinausgeht. Das Relokations-Problem ist somit vollkommen beseitigt.

Das Link-Programm ist nicht ganz so einfach zu lösen, weil die relative Lage zweier Module im Speicher erst zur Laufzeit bekannt ist. Wenn das Modul A eine Prozedur aufrufen oder zu einer Variablen zugreifen muß, die in Modul B enthalten ist, muß es eine Möglichkeit haben, die absolute Adresse der externen Prozedur oder Variablen zu ermitteln. Bei den meisten Mikroprozessoren ist die einzige Möglichkeit, dies zu tun, die Verwendung eines Link-Editors, der durch den Code jedes Moduls geht und jede externe Referenz mit der korrekten absoluten Adresse versorgt.

Als Direktor für Product Marketing Europe, Mikroprozessoren und Systeme, ist **Dr. Werner Trattnig** verantwortlich für die Leitung der Marketing-Unternehmen in Europa für die Prozessorfamilie 32000. Gleichzeitig leitet er das Marketing-Programm für OEM-Computer-Produkte, Software und Entwicklungssysteme. Bevor Dr. Trattnig zu National Semiconductor kam, war er Professor für Computerwissenschaften und Elektrotechnik an der Stanford Universität bei San Francisco, Kalifornien. Er leitete Entwicklungsprojekte für höhere Computer-Architekturen und Programmiermethoden. Dr. Trattnig stammt aus Österreich und promovierte an der technischen Universität Wien zum Doktor der Technischen Wissenschaften und an der Universität Wien zum Doktor der Wirtschaftswissenschaften. Nach Abschluß seiner Doktorarbeiten entwickelte er am Massachusetts Institute of Technology, USA, Multiprozessorsysteme.



Die 32000-Architektur enthält spezielle Register, Adressierungsarten und Befehle, die die Kommunikation zwischen Modulen erleichtern. Der Basis-Mechanismus für die Unterstützung von Software-Modulen ist die Modul-Tabelle, die für jedes Modul einen 16-Bit-Modul-Deskriptor enthält. Alle Programme bestehen aus drei Abschnitten:

- einem Code-Abschnitt, der den Programm-Code und die Modul-Konstanten (Literale) enthält,
- einem statischen Daten-Abschnitt, der die globalen Variablen und Daten des Moduls enthält (in einem Pascal-Programm z. B. würden die im äußersten Block deklarierten Daten-Strukturen gespeichert),
- einer Link-Tabelle, die die absoluten Adressen der im Modul verwendeten externen Variablen und Prozeduren enthält (weil alle externen Referenzen über die Link-Tabelle gemacht werden können, braucht der Code-Abschnitt nie absolute Adressen zu enthalten).

Beim Laden jedes Moduls füllt der Linking-Loader einfach die entsprechenden Einträge in der Modul-Tabelle mit den absoluten Adressen des Code-Abschnitts, des statischen Daten-Abschnitts und der Link-Tabelle. Wenn allen Modulen absolute Adressen zugewiesen sind, füllt der Linker die Link-Tabellen mit den richtigen Adressen. In den Code-Abschnitten sind keinerlei Änderungen erforderlich, so daß Module unabhängig voneinander im ROM gespeichert und zum System hinzugefügt werden können. Auch können die 32-Bit-Pointer in der Modul-Tabelle jeden Punkt innerhalb des Adreßraumes erreichen, so daß Module überall im Speicher untergebracht werden können.

Uwe Krause

Höhere Leistung und besserer Software-Schutz:

Slave-Prozessoren in 32000-Systemen

Obwohl Slave-Prozessoren meistens im Zusammenhang mit Gleitkomma-Operationen und Speicherverwaltungs-Aufgaben eingesetzt werden, können sie auch einer Vielzahl anderer Funktionen dienen. Einige bedeutende Anwender der 32000-Familie z. B. setzen Slave-Prozessoren dafür ein, hohe Software-Entwicklungsinvestitionen zu schützen. Die Implementierung einiger wichtiger Betriebssystem-Funktionen in die Hardware eines kundenspezifischen Slave-Prozessors kann nicht nur die Leistungsfähigkeit des Systems vergrößern, sondern führt auch dazu, daß es für Konkurrenten wesentlich schwieriger ist, das jeweilige Produkt oder System zu kopieren und einfach nachzubauen.

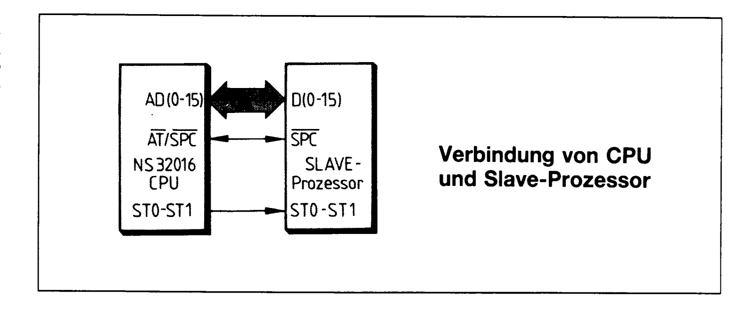
Bei vielen Mikroprozessoren ist die Entwicklung von kundenspezifischen Slave-Prozessoren, die zuverlässig mit dem Hauptprozessor zusammenarbeiten, eine sehr schwierige Aufgabe. Ein spezielles Merkmal der 32000-Architektur ist die Berücksichtigung von kundenspezifischen Slave-Prozessoren. Die Schnittstellen-Protokolle sind genau definiert und ausführlich dokumentiert. Innerhalb des Befehlssatzes gibt es zwanzig Instruktionen, die für den Einsatz von kundenspezifischen Slave-Prozessoren reserviert sind.

Bevor ein Slave-Prozessor in einem 32000-System arbeiten kann, muß dem Hauptprozessor dessen physikalische Präsenz mitgeteilt werden. Dies geschieht gewöhnlich während der Systeminitialisierung nach dem Reset durch Setzen eines dafür vorgesehenen Bits im Konfigurationsregister der CPU. Die „CFG"-Bits haben folgende Bedeutung:

Das „I"-Bit zeigt an, ob ein 32202-Interrupt-Controller vorhanden ist oder nicht. Falls vorhanden, sind die Interrupt-Anforderungen an dem INT-Anschluß vektorisiert, im anderen Falle – wenn kein Interrupt-Controller vorhanden ist – sind sie nicht vektorisiert.

Die „F"-, „M"- und „C"-Bits zeigen an, ob ein Gleitkomma-Prozessor, eine Speicherverwaltungs-Einheit oder ein kundenspezifischer Slave-Prozessor eingebaut sind oder nicht.

Falls die CPU einen Slave-Prozessor-Befehl empfängt und das zugehörige „CFG"-Bit ist nicht gesetzt, tritt ein undefinierter Befehlszustand auf (Undefined-Instruction-Trap). Bei richtigem Gebrauch des Trap-Mechanismus kann ein physikalisch nicht vorhandener Slave-Prozessor durch Software emuliert werden. Dies ist dann hilfreich, wenn aus Kosten-/Leistungsgründen oder wegen der laufenden Hardware-Entwicklung noch kein Slave-Prozessor im System ist. Wenn das entsprechende „CFG"-Bit gesetzt ist, so geht die CPU davon aus, daß der zugehörige Slave-Prozessor vorhanden ist.



Verbindung von CPU und Slave-Prozessor

Die vom Slave-Prozessor ausführbaren Funktionen werden ausschließlich vom Benutzer festgelegt. Entsprechend haben die Slave-Prozessor-Instruktionen, die Teil des Befehlssatzes der 32000er-Serie sind, keine festgelegten Funktionen. Es handelt sich dabei um Operationscode-Kombinationen, wobei der Hersteller garantiert, daß diese Kombinationen nicht in zukünftigen CPUs benutzt werden, so daß die Auf- und Abwärtskompatibilität innerhalb der 32000-Familie in gleichem Maße für die kundenspezifischen Slave-Prozessoren gilt.

Die Operationscode-Kombinationen wurden gewählt, um verschiedene praktische Befehlsformate bereitzustellen, die kompatibel zu den Arten von Operationen sind, die der Benutzer gewöhnlich ausführen möchte. *Tabelle 1* enthält die zur Verfügung stehenden Befehle.

Die Kennzeichnung „c“ wird benutzt, um anzuzeigen, daß der Operand ein 32-Bit-Doppelwort (d) oder ein 64-Bit-Vierfachwort (Q) sein kann. Entsprechend steht „i“ für ein Byte, Wort oder Doppelwort (ganzzahlig). In beiden Fällen ist das gültige Format durch einen Mnemonic-Zusatz gekennzeichnet.

Für einen guten Bedienungskomfort sind den Instruktionen neutrale Mnemonics zugeordnet. Die CCAL-Befehle entsprechen z. B. in der Weise den Gleitkomma-Rechenbefehlen, daß zwei Operanden gelesen werden und das Ergebnis dann zurück auf den zweiten Operanden geschrieben wird. Anzumerken ist, daß die reinen Mnemonics nicht eindeutig sind. Der „Custom-Compare“-Befehl (CCMP), z. B., ähnelt sehr dem Fließkomma-„Compare“, in dem zwei Operanden gelesen werden und die N-, Z- und L-Bits im CPU-Prozessor-Statusregister von diesem Befehl geändert werden. Die Operation des Slave-Prozessors als Antwort auf einen CCMP-Befehl muß jedoch nicht tatsächlich eine Vergleichsoperation sein.

Befehlsprotokoll

Obwohl der Anwender vollständig frei in der Entscheidung ist, wie der kundenspezifische Slave-Prozessor seine Befehle interpretieren soll, ist die Art, in der die Befehle und Daten zum Slave-Prozessor geleitet werden, präzise festgelegt. Der Slave-Prozessor muß mit bestimmten CPU-Steuerleitungen und dem Adressen-/Daten-Bus verbunden sein und mit der CPU auf der Basis des Standard-Slave-Protokolls kommunizieren.

Bild 1 zeigt die physikalische Verbindung eines Slave-Prozessors mit einer CPU der 32000-Familie. ST0 und ST1 sind die beiden CPU-Statusleitungen mit der geringsten Wertigkeit und werden zur Identifizierung der unterschiedlichen Protokollstufen benutzt. Befehle und Operanden werden dem Prozessor im Buszyklus-Takt zugeleitet. Während des Buszyklusses nutzt die CPU die SPC-Leitung wie einen aktiven Low-Data-Impuls. SPC ist ein Signal für zwei verschiedene Zwecke, da es auch vom Slave-Prozessor gesendet wird, um die Vollständigkeit einer Operation mitzuteilen. Der Buszyklus in den 32016- und 32032-Systemen arbeitet mit Byte- oder Wort-Transfers. Die CPU 32332 unterstützt zusätzlich 32-Bit-Slave-Transfers.

Der erste Schritt in das Protokoll bei einem System mit Fließkomma-Prozessor, Speicherverwaltungs-Einheit und kundenspezifischen Slave-Prozessoren ist die Herstellung einer Verbindung mit dem jeweiligen Slave-Prozessor. Alle Slave-Prozessor-Befehle haben ein Basis-Befehlsfeld von 3 Byte. Das erste Byte ist ein ID-Byte. Dieses weist einen Befehl als Slave-Instruktion aus, gibt an, welcher Slave-Prozessor den Befehl auszuführen hat und bestimmt das Format des folgenden Operationswortes.

Wenn die CPU einen Befehl als Slave-Prozessor-Instruktion identifiziert, sendet sie das ID-Byte auf die

Tabelle 1. Slave-Befehlsprotokolle (Auszug aus Original-Dokumentation)

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issued	Returned Value Type and Dest.	PSR Bits Affected
CCAL0c	read.c	mw.c	c	c	c to Op. 2	none
CCAL1c	read.c	mw.c	c	c	c to Op. 2	none
CCAL2c	read.c	mw.c	c	c	c to Op. 2	none
CCAL3c	read.c	mw.c	c	c	c to Op. 2	none
CMOV0c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV1c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV2c	read.c	write.c	c	N/A	c to Op. 2	none
CCMPc	read.c	read.c	c	c	N/A	N,Z,L
CCV0ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV1ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV2ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV3ic	readi	write.c	i	N/A	c to Op. 2	none
CCV4DQ	read.D	write.Q	D	N/A	Q to Op. 2	none
CCV5QD	read.Q	write.D	Q	N/A	D to Op. 2	none
LCSR	read.D	N/A	D	N/A	N/A	none
SCSR	N/A	write.D	N/A	N/A	D to Op. 2	none
CATST0*	addr	N/A	D	N/A	N/A	F
CATST1*	addr	N/A	D	N/A	N/A	F
LCR*	read.D	N/A	D	N/A	N/A	none
SCR*	write.D	N/A	N/A	N/A	D to Op. 1	none

D = Doppelwort
i = Integer-Format (BWD) spezifiziert im Mnemocode
f = Fließkomma-Format (F, L) spezifiziert im Mnemocode
N/A = Nicht anwendbar für diese Anwendung

niedrigwertige Hälfte des Datenbusses und setzt den Statuscode (ST0, ST1) auf 11. Alle angeschlossenen Slave-Prozessoren empfangen das ID-Byte und decodieren es. Der Slave-Prozessor, der durch das ID-Byte ausgewählt ist, wird aktiviert. Alle anderen Slave-Prozessoren innerhalb des Systems sind nun an der Bearbeitung des restlichen Protokolls nicht mehr beteiligt. Für die Slave-Prozessoren hat das ID-Byte einen der hexadezimalen Werte $16, $36 oder $B6, abhängig vom Format des Befehles.

In der zweiten Protokollstufe, angezeigt durch den Statuscode 01, sendet die CPU zum Slave-Prozessor das Operationswort und alle erforderlichen Operanden. Das Operationswort wird zuerst gesendet und muß vom Slave-Prozessor decodiert werden, so daß sowohl die CPU als auch der Slave-Prozessor wissen, wieviele Operanden folgen werden. Die Adressierung der Operanden kann in allen allgemeinen Adressierarten erfolgen, aber der Aufruf der Operanden geschieht durch die CPU. Der Slave-Prozessor wartet einfach auf die ihm übermittelten Werte und beginnt dann mit der Ausführung des Befehls.

Wenn der Slave-Prozessor eine Operation abgeschlossen hat, signalisiert er der CPU dies durch ein Low-Signal auf der SPC-Leitung. Die SPC-Leitung wird für diesen Zweck durch einen 5-kΩ-Widerstand in der CPU auf High-Pegel gehalten. Wenn die SPC-Leitung durch das entsprechende Slave-Prozessor-Signal auf Low-Pegel geht, setzt die CPU den Statuscode auf 10 und liest über die SPC-Leitung ein Statuswort aus dem Slave-Prozessor. Mit Ausnahme der CCMP-, CATST0- und CATST1-Instruktionen, die einige PSR-Flags ändern, hat nur das geringstwertige Bit des Slave-Statuswortes eine Bedeutung. Das LSB kann dazu benutzt werden, einen Fehler in der Slave-Funktion anzuzeigen.

Zum Schluß setzt die CPU den Statuscode wieder auf 01 und liest alle Ergebnisse des Slave-Prozessors. Die *Tabelle* faßt die einzelnen Stufen des Protokolls zusammen. An dieser Stelle ist anzumerken, daß die Architektur der 32000-Familie besser mit Slave-Prozessoren arbeitet als mit konkurrierenden Coprozessoren, weil die FPU und die MMU so schnell sind, daß die Haupt-CPU keine Leistungsverbesserung im Sinne einer nützlichen Arbeit erführe, wenn sie in den paar Mikrosekunden die z. B. eine Fließkomma-Multiplikation benötigt, zur Verfügung stände. Die einzige Arbeit, die die CPU während der Befehlsausführung des Slave-Prozessors machen kann, ist das Zurückholen von Befehlen in die Warteschlange. Falls die Warteschlange voll ist, bevor der Slave-Prozessor seinen Befehl ausgeführt hat, wartet die CPU, bis die Ausführung signalisiert ist.

Tabelle 2: Protokoll für die Coprozessor-Kommunikation

Schritt	Status	Aktion
1	11	CPU sendet ID-Byte
2	01	CPU sendet Operationswort
3	01	CPU sendet erforderliche Operanden
4	XX	Slave-Prozessor beginnt mit der Befehlsausführung
5	XX	Slave-Prozessor schaltet SPC-Leitung auf Niedrigpegel
6	10	CPU liest Status-Wort
7	01	CPU liest Ergebnisse, fals angefordert

Hochsprachengerechter Befehlssatz

Damit Hochsprachen so effizient wie möglich unterstützt werden, ist der Befehlssatz der 32000-Familie vollständig symmetrisch und orthogonal. So kann jeder Befehl mit jedem richtigen Datentyp genutzt werden und auf Operanden läßt sich in jedem Adreßmodus zugreifen. Weil die Zahl der Operationscode-Kombinationen, die erforderlich sind, um alle möglichen Kombinationen von Befehlen, Datentypen und Adressierarten zu behandeln, die Zahl derjenigen übertrifft, die durch ein 16-Bit-Befehlswort spezifizierbar sind, haben die 32000-Befehle ein variables Format.

Das Diagramm zeigt das allgemeine Format eines Befehls. Der Basisbefehl ist 1 bis 3 Byte lang und enthält den Operationscode und ein oder zwei Felder, jedes bis zu 5 Bit lang, die den allgemeinen Operanden-Adressierungsmodus festlegen. Abhängig von dem speziellen Befehl und der Art, in der auf die Operanden zugegriffen wird, können dem Basisbefehl ein oder mehrere Erweiterungs-Bytes folgen.

Wenn z. B. der leistungsfähige skalierte Index-Modus benutzt wird, um auf einen oder auf beide der Operanden zuzugreifen, enthält das allgemeine Adressierungsarten-Feld in dem Basisbefehl den Skalierfaktor (1, 2, 4 oder 8), und ein oder zwei zusätzliche Index-Bytes sind darüber hinaus erforderlich, um anzugeben, welche Register als Indexregister benutzt werden sollen und welche Adressierungsart anzuwenden ist, bevor die Indizierung stattgefunden hat.

Die Adressierarten erlauben die zusätzliche Angabe von relativen Adressen. Der Speicher-Relativ-Modus z. B. enthält zwei relative Adressen. Die erste wird dem Inhalt eines der für spezielle Zwecke verwendeten Register hinzugefügt (Stack-Pointer, Frame-Pointer, Static-Base oder Program-Counter), was die Adresse eines zweiten Zeigers ergibt. Die zweite relative Adresse wird der neuen Basisadressen hinzugefügt, um die endgültige Operandenadresse zu bestimmen.

Um die Befehlswort-Länge so kurz wie möglich zu halten, wird das Format einer relativen Adresse innerhalb der obersten Bit eines Feldes codiert. Eine relative Byte-Adresse, gekennzeichnet durch ein MSB von 0, belegt den 7-Bit-Bereich von −64 bis +63. Wenn das MSB 1 ist, so legt das nächste MSB fest, ob die relative Adresse eine Wort-Adresse (NMSB = 0), die den Bereich von −8192 bis +8191 belegt, oder eine Doppelwort-Adresse (NMSB = 1), die mehr als den gesamten 16-MByte-Adreßbereich der CPU umfaßt, ist.

Mike Hudson

Fließkomma-Peripherieeinheit mit der FPU NS32081

Fließkommapakete, die jeweils ein spezielles Format zur Darstellung von Zahlen haben, waren in der Vergangenheit für Besitzer von Kleincomputern eine Quelle dauernden Ärgers. Seit 1977 arbeitete das IEEE-Standardisierungskomitee an der Entwicklung eines genormten Verfahrens für binäre Fließkomma-Arithmetik. Die Standardvorschläge spezifizieren zwei grundsätzliche Fließkommaformate: 32 Bit (einfache Genauigkeit) und 64 Bit (doppelte Genauigkeit), außerdem Konvertierungen zwischen verschiedenen Zahlendarstellungen und eine Mindestausstattung an Befehlen wie z. B. Addition, Subtraktion, Multiplikation, Division und Ziehen von Quadratwurzeln. Bei der FPU (Floating Point Unit) NS32081 handelt es sich um einen Slave-Prozessor, der als Hochgeschwindigkeits-Hardware-Implementierung die Fließkommaoperationen nach dem IEEE-Standard im Befehlssatz der Mikroprozessoren aus der 32000-Familie bearbeitet. Beim derzeitigen Stand der Halbleitertechnologie ist es noch nicht sinnvoll, alle Funktionen des IEEE-Standards zu implementieren, so daß man beim NS32081 nur einen Teil der Normvorschläge verwirklicht hat. Operationen, wie z. B. Ziehen von Quadratwurzeln, erfordern daher zusätzliche Software.

Obwohl der Baustein zunächst für den Betrieb in Zusammenhang mit 32000-Systemen konzipiert war, lassen es Geschwindigkeit und Flexibilität zu, diese FPU auch in Zusammenhang mit anderen Mikroprozessoren zu betreiben. Der Zweck dieses Beitrages ist unter anderem, zu zeigen, wie dies geschieht. Davor ist es allerdings notwendig, die Betriebsweise der FPU in einem System näher zu beleuchten.

Programmiermodell

Die FPU ist vollständig in die 32000-Architektur integriert und Fließkommazahlen werden durch Zwei-Adreß-Befehle der 32000-CPU behandelt, wobei neun Spezialregister benutzt werden. Die Register sind im FPU-Baustein implementiert, allerdings greift dieser nicht direkt auf den Systemspeicher zu. Statt dessen interpretiert die CPU den Befehl und holt die Operanden, so daß sie diese zusammen mit dem Kommandocode an die FPU weitergeben kann. Dies bedeutet, daß die FPU auch dazu benutzt werden kann, den Durchsatz von weniger leistungsfähigen CPUs zu erhöhen. Dazu ist es notwendig, daß die richtige Reihenfolge bei der Präsentierung der Kommandos und Daten beim Schreiben und Lesen eingehalten wird.

Bild 1 zeigt die neun Register, auf die ein Benutzer zugreifen kann. Die acht 32-Bit-Datenregister erlauben einen schnellen Zugriff auf acht einfache oder vier doppelt präzise Fließkomma-Operanden. Wenn Operanden doppelter Genauigkeit benutzt werden, beinhaltet das geradezahlige Register die niederwertige Hälfte des Operanden.

Das Fließkomma-Statusregister (FSR) ist in *Bild 2* detaillierter dargestellt. Die oberen 16 Bit sind für

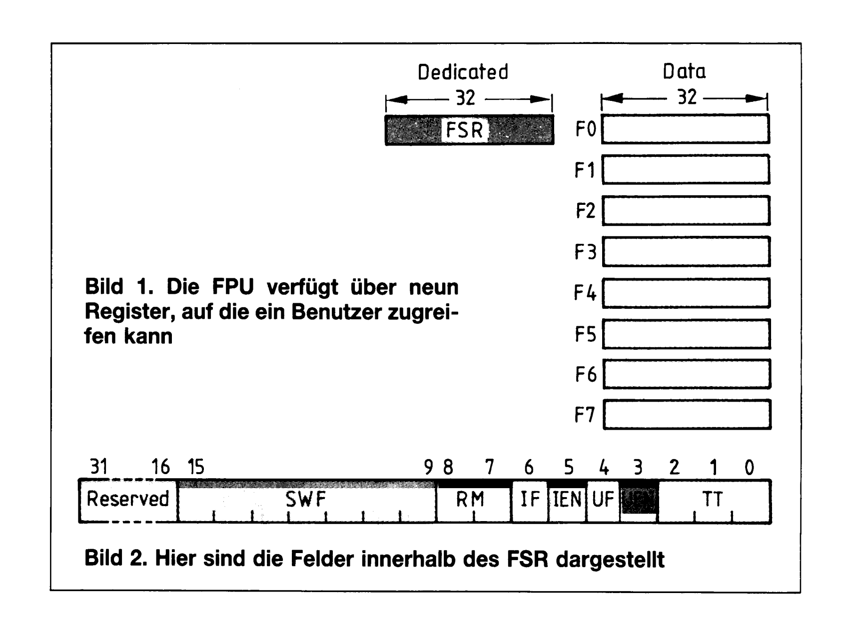


Bild 1. Die FPU verfügt über neun Register, auf die ein Benutzer zugreifen kann

Bild 2. Hier sind die Felder innerhalb des FSR dargestellt

zukünftige Zwecke reserviert, während die untere Hälfte in folgende drei Felder aufgeteilt sind:

1. Mode Control Feld

Die Bits 7 und 8 legen die Methode fest, mit der Resultate gerundet werden. Resultate können auf die nächste ganze Zahl, auf Null, gegen Positiv unendlich oder gegen Negativ unendlich gerundet werden.

Bit 3 bestimmt die Auswirkung des sogenannten „underflows“, das heißt ein Resultat, das als Absolutwert zu klein ist, um als normierte Zahl dargestellt zu werden. Wenn dieses Bit gesetzt ist, veranlaßt die FPU einen Software-Interrupt, sonst gibt eine „underflow“-Bedingung ein Ergebnis von genau Null. In ähnlicher Weise bestimmt Bit 5 die Auswirkung eines „overflow“ Ergebnisses. Wenn Bit 5 gesetzt ist, wird ein Software-Interrupt veranlaßt, sonst wird der Wert entsprechend der Anweisung in Bit 7 und 8 gerundet.

die Formate 9 und 11 des Befehlssatzes für den 32000. *Bild 3* zeigt die Struktur dieser beiden Befehlsformate. In beiden Fällen spezifiziert das Opcode-Feld die betreffende Fließkommaoperation, die auszuführen ist. Die Felder i und f benutzt man zur Auswahl der Datentypen. Ganzzahlige Werte können eine Länge von Byte, Wort- oder Doppelwort annehmen und Fließkommazahlen können einfache oder doppelte Genauigkeit aufweisen.

Tabelle 1 stellt die 16 Fließkommaoperationen zusammen, die sich vom Opcode-Feld spezifizieren lassen. Neben den standardmäßigen Arithmetikfunktionen gibt es leistungsfähige Move- und Umsetzeinrichtungen sowie die Pascal-Operationen Round, Trunc und Floor.

Die Felder mit den Labels gen 1 und gen 2 werden normalerweise bei 32000-Befehlen zur Spezifizierung von Quellen- und Bestimmungs-Operanden bei allgemeinen Adressierarten verwendet. Allerdings führt die FPU keine Speicherzugriffe aus, so daß die Notwendig-

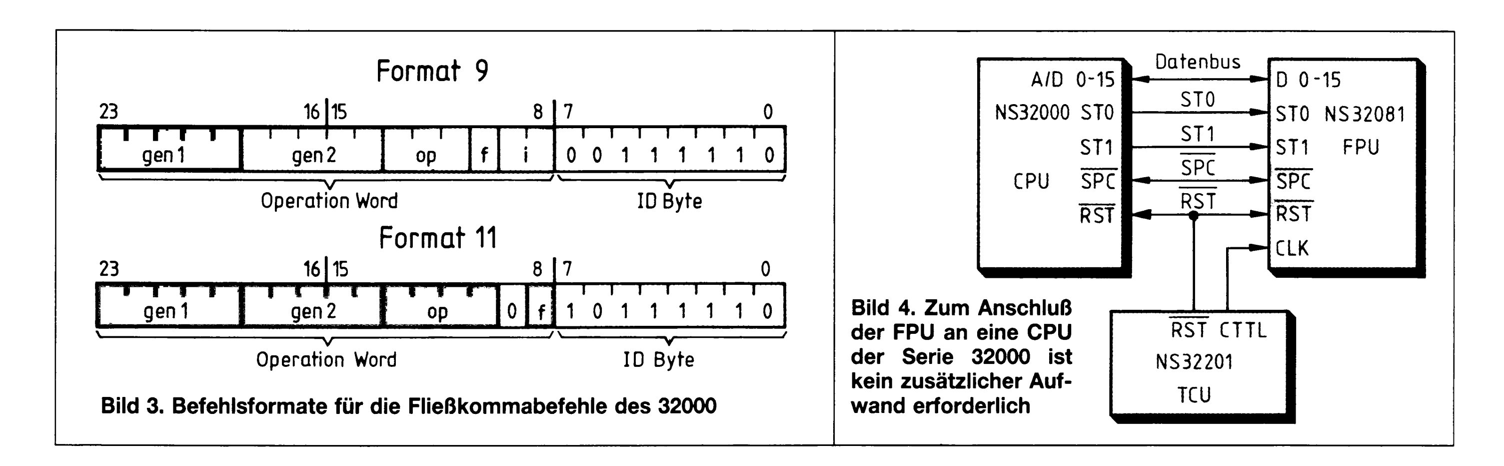


Bild 3. Befehlsformate für die Fließkommabefehle des 32000

Bild 4. Zum Anschluß der FPU an eine CPU der Serie 32000 ist kein zusätzlicher Aufwand erforderlich

2. Statusfeld

Bits 4 und 6 sind Flag-Bits, die ein „underflow“ oder „overflow“ Resultat anzeigen. Sie werden nicht von Bit 2 und 5 beeinflußt. Bits 0...2 werden dazu benutzt, anzuzeigen, welche Ursache es hatte, daß die FPU eine Software Interrupt veranlaßt hat.

3. Software-Feld

Bits 9...15 des Statusregisters werden nicht von der FPU-Schaltung benutzt, doch lassen sich hier Informationen von der CPU ablegen. Diese Bits werden auch von der Fließkomma-Erweiterungssoftware benutzt, wenn allerdings die FPU in Zusammenhang mit einem anderen Prozessor Anwendung findet, stehen diese Bits dem Programmierer zur Verfügung.

Befehlsprotokoll

Im Befehlssatz der Serie 32000 haben die Befehle des Slave-Prozessors ein 3-Byte-Grund-Befehlsfeld, das aus dem ID-Byte und einem Operationswort besteht. Das ID-Byte identifiziert den Befehl als eine Instruktion des Slave-Prozessors, spezifiziert, welcher Slave-Prozessor den Befehl ausführt und bestimmt das Format des folgenden Operationswortes. Fließkommabefehle belegen

keit zur Benutzung dieser Felder mit anderer CPU bei der Berechnung von Operanden-Adressen nicht gegeben ist. Statt dessen spezifiziert ein Wert zwischen 0 und 7 eines der Fließkommaregister F0...F7 als Platz des entsprechenden Operanden, während bei jedem Wert über 7 angezeigt wird, daß der Operand von der CPU als Teil des Befehlsprotokolls transferiert wird. Alle Nicht-Fließkomma-Operationen müssen auf diese Weise transferiert werden, weil nur Fließkommazahlen im Register-File der FPU untergebracht sein können.

Bild 4 zeigt die physikalische Verbindung zwischen der FPU und der CPU aus der 32000-Familie. Wenn eine andere CPU benutzt wird, müssen die Signale ST0, ST1, $\overline{\text{SPC}}$ und $\overline{\text{RST}}$, die normalerweise direkt von der 32000-CPU zur Verfügung stehen, aus anderen CPU-Signalen abgeleitet werden. Befehle und Operanden werden mit Hilfe von Slave-Prozessor-Buszyklen zur FPU weitergeleitet. Während dieser Buszyklen benutzt die CPU die $\overline{\text{SPC}}$-Leitung als ein Daten-Strobe-Signal (aktiv auf Low-Pegel), während ST0 und ST1 die verschiedenen Stufen des Protokolls definieren. Es ist zu beachten, daß $\overline{\text{SPC}}$ ein Signal mit zwei Aufgaben ist, weil mit dessen Hilfe der Slave-Prozessor auch den Abschluß einer Operation anzeigen kann.

Weil die Architektur des 32000 bis zu vier verschiedene Slave-Prozessoren gleichzeitig unterstützen kann, ist es der erste Schritt im Protokoll, die Kommunikation zum betreffenden Slave-Prozessor aufzubauen. Wenn die CPU einen Befehl als Slave-Prozessor-Instruktion identifiziert, überträgt sie das ID-Byte auf die niedrigwertige Hälfte des Datenbusses und setzt den Status-Code (ST0, ST1) auf 11. Alle angeschlossenen Slave-Prozessoren bekommen das ID-Byte und decodieren dieses. Der Slave-Prozessor, der vom ID-Byte ausgewählt wurde, wird aktiviert, alle anderen Slave-Prozessoren im System nehmen bei der Übertragung des Protokolls nicht teil.

In der zweiten Stufe des Protokolls, die durch den Status-Code 01 angezeigt wird, sendet die CPU und die FPU das Operationswort sowie die erforderlichen Operationen. Das Operationswort wird zuerst gesendet, wobei die hochwertigen und niederwertigen Bytes auf dem Datenbus miteinander vertauscht sind. Diese werden von der FPU decodiert, so daß FPU und CPU wissen, wie viele Operanden folgen. Operanden lassen sich mit Hilfe jeder allgemeinen Adressierart adressieren. Auf diese Weise werden Fließkommazahlen genauso behandelt wie jede beliebige andere numerische Datentype in 32000-Systemen. Das Aufnehmen der Operanden wird allerdings durch die CPU vorgenommen. Die FPU wartet ganz einfach solange, bis die tatsächlichen Werte zu ihr übertragen werden und startet anschließend das Ausführen des Kommandos.

Wenn die FPU die Operation beendet hat, signalisiert sie dies der CPU, indem $\overline{\text{SPC}}$ auf Low geschaltet wird. Diese Leitung wird normalerweise mit Hilfe eines 5-kΩ-Widerstandes auf High-Pegel gehalten. Die CPU setzt daraufhin den Status-Code auf 10 und benutzt $\overline{\text{SPC}}$ zum Lesen des Statuswortes aus der FPU. Das Statuswort enthält neben anderem ein Flag, das anzeigt, ob die FPU einen Fehler entdeckt hat; wenn das Flag-Bit gesetzt ist, bricht die CPU das Protokoll ab und springt über den FPU-Vektor in die Interrupt-Tabelle.

Zum Schluß setzt die CPU den Status-Code wieder auf 01 und liest die von der FPU stammenden Resultate. *Tabelle 2* faßt die Stufen des Protokolls zusammen. Um die FPU mit einem anderen Mikroprozessor zu verbinden, muß natürlich ein ST0-, ST1- sowie $\overline{\text{SPC}}$-Signal erzeugt werden, und außerdem dafür gesorgt werden, daß Kommandos sowie Operanden der FPU in den korrekten Formaten vorliegen.

Interface für 68000

Bild 5 ist die Blockschaltung, die zum Anschluß der FPU an den Mikroprozessor MC68000 erforder-

Tabelle 1. Fließkomma-Operationen der FPU

Format	Op	Befehle	Beschreibung
11	0001	MOVf gen1, gen2	move without conversion
9	010	MOVLF gen1, gen2	move, converting from double to single precision
9	011	MOVFL gen1, gen2	move, converting from single to double precision
9	000	MOVif gen1, gen2	move, converting from any integer type to any floating point type
9	100	ROUNDfi gen1, gen2	move, converting from floating point to nearest integer
9	101	TRUNCfi gen1, gen2	move, converting from floating point to nearest integer closer to zero
9	111	FLOORfi gen1, gen2	move, converting from floating point to largest integer less than or equal
11	0000	ADDf gen1, gen2	add gen1 to gen2
11	0100	SUBf gen1, gen2	subtract gen1 from gen2
11	1100	MULf gen1, gen2	multiply gen2 by gen1
11	1000	DIVf gen1, gen2	divide gen2 by gen1
11	0101	NEGf gen1, gen2	move negative of gen1 to gen2
11	1101	ABSf gen1, gen2	move absolute value of gen1 to gen2
11	0010	CMPf gen1, gen2	compare gen1 to gen2
9	001	LFSR gen1	load FSR
9	110	SFSR gen2	store FSR

Tabelle 2. Protokoll-Schritte

Schritt	Status	Aktion
1	11	CPU sendet ID-Byte
2	01	CPU sendet Operations-Wort
3	01	CPU sendet die erforderlichen Operanden
4	XX	FPU beginnt mit der Ausführung
5	XX	FPU schaltet SPC auf Low
6	10	CPU liest Statuswort
7	01	CPU liest, wenn erforderlich, Resultate

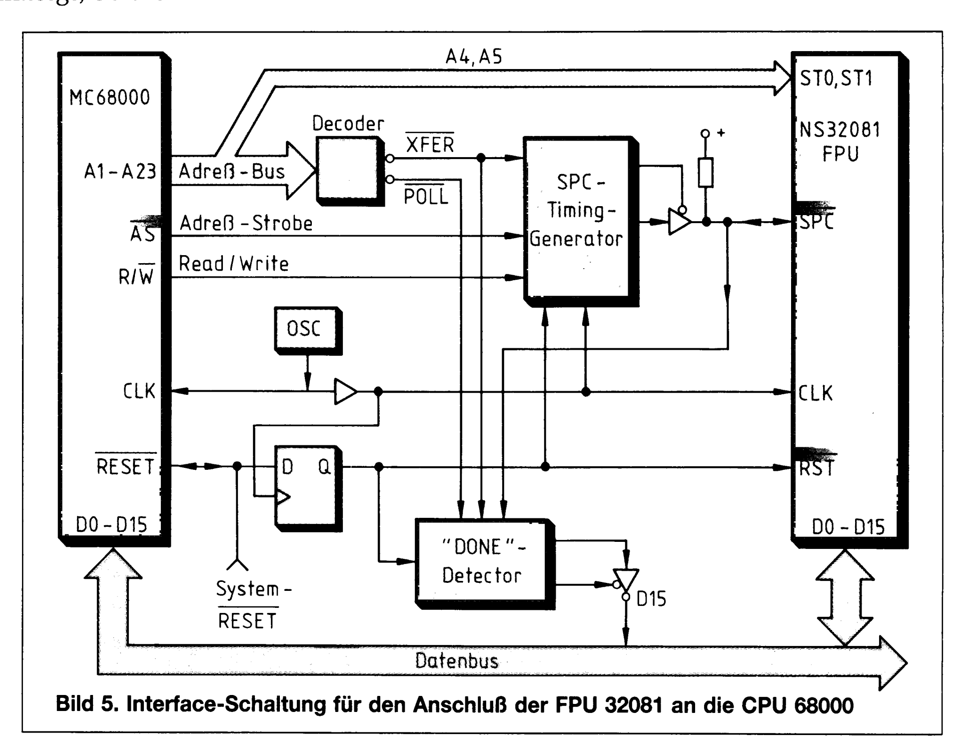


Bild 5. Interface-Schaltung für den Anschluß der FPU 32081 an die CPU 68000

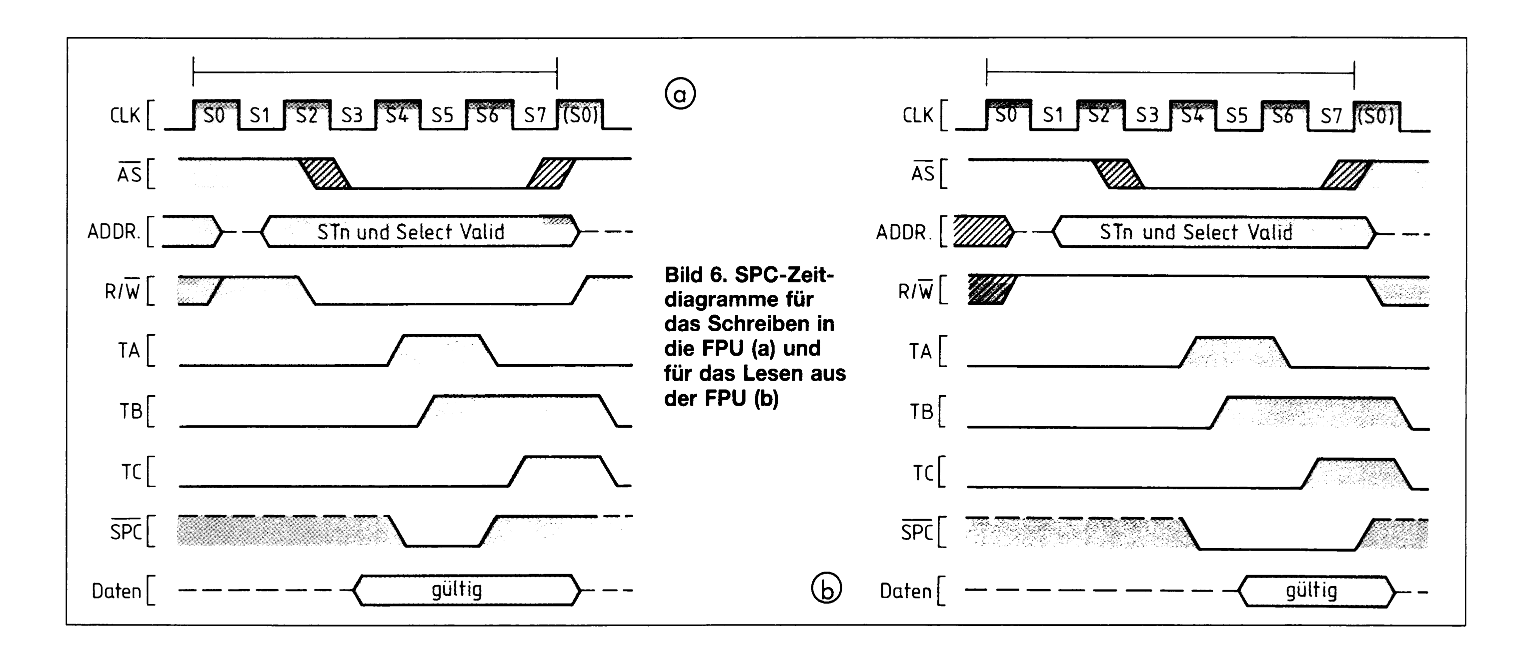


Bild 6. SPC-Zeitdiagramme für das Schreiben in die FPU (a) und für das Lesen aus der FPU (b)

lich ist. Direkte Verbindungen sind beim Datenbus und den Taktanschlüssen möglich, außerdem für die Signale ST0, ST1, die ganz einfach von den Adreßleitungen A4, A5 des Adreß-Busses vom 68000 abgeleitet werden. Die zusätzliche Schaltung ist allerdings für das Reset-Signal sowie die SPC-Leitung erforderlich. Das Reset-Signal des 68000 muß mit dem Takt synchronisiert werden, bevor es am RST-Anschluß der FPU anliegen kann, während das SPC-Signal kein entsprechendes Signal beim 68000 hat und daher aus dem Adreß-Strobe sowie den Schreib-/Lese-Signalen abgeleitet werden muß.

Die Beziehung zwischen SPC und den Signalen des 68000 zeigt *Bild 6*. Die Schaltung, mit der das SPC-Signal erzeugt wird, ist in *Bild 7* dargestellt. Wegen der zwei Funktionen des SPC-Signals erkennt der Adreß-Decodierer (Bild 5) zwei separate Adreßbereiche: Einer ist für die Übertragung der Informationen von der FPU erforderlich, während der andere von dem 68000 benutzt wird, um die DONE-Impulse der FPU abzufragen. Der Adreß-Decodierer besitzt Ausgänge mit den Bezeichnungen XFER und POLL (Bild 5).

Bild 6*a* zeigt die Zeitsteuersignale, die erzeugt werden, wenn der 68000 in die FPU schreibt. Der SPC-Anschluß ist auf undefiniertem Potential gehalten (oder mit einem Pullup-Widerstand auf High-Pegel gezogen) bis der Buszustand S4 erreicht wird, das ist der Zeitpunkt, bei dem dieser Punkt auf Low-Pegel gezogen wird. Mit der nächsten ansteigenden Flanke von CLK wird SPC aktiv auf High-Pegel gezogen und anschließend hochohmig geschaltet. Aktives Anlegen eines Potentials ist erforderlich, weil diese Leitung unakzeptabel lange Anstiegszeiten über 4 MHz aufweist. Die Zeitgeberkette (Bild 7) erzeugt die Signale TA, TB und TC, die Zustände und Enable-Steuerungen der SPC-Leitung übernehmen.

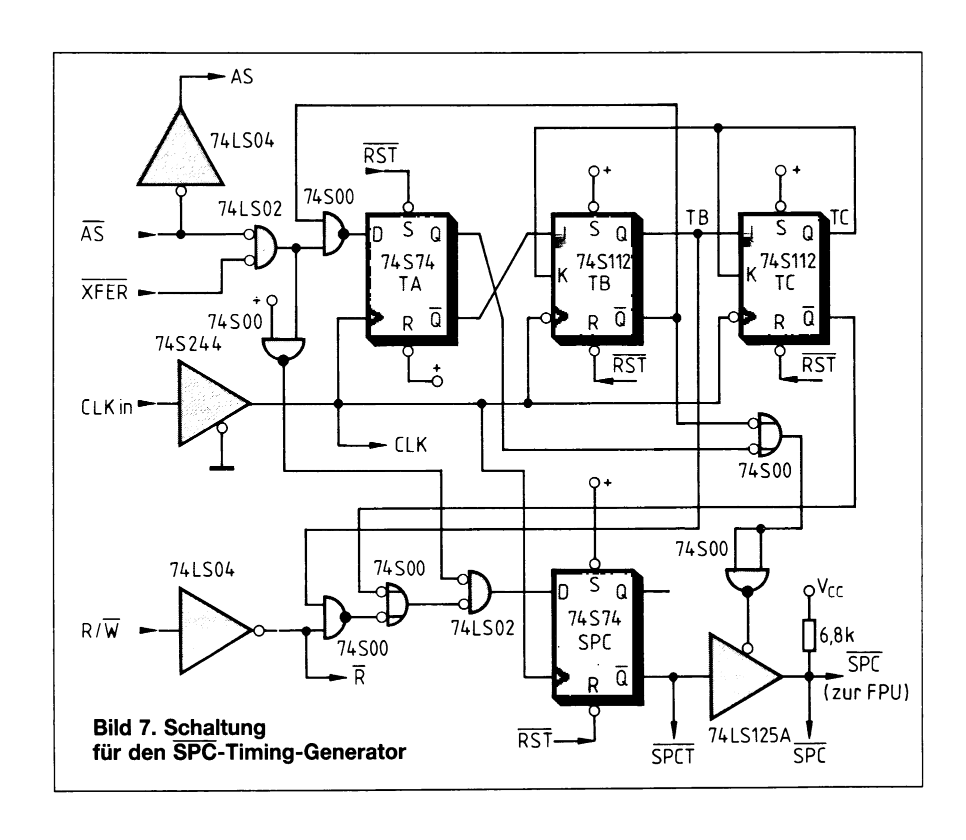


Bild 7. Schaltung für den SPC-Timing-Generator

Bild 6*b* zeigt das SPC-Timing, wenn die CPU 68000 aus der FPU liest. Der wichtigste Unterschied dabei ist, daß SPC für zwei Taktperioden aktiv bleibt, so daß die FPU Daten auf dem Bus hält, bis diese vom 68000 gelesen sind. Wie schon im vorhergehenden Fall wird SPC aktiv auf High-Pegel gelegt, bevor dieser Anschluß freigegeben wird, um sicherzustellen, daß es keine Probleme mit der Anstiegszeit gibt. Es ist zu beachten, daß SPC

nicht mehr als zwei Taktperioden nach der vorhergehenden abfallenden Flanke angesteuert werden darf, weil die FPU nicht schneller als drei Taktperioden nach einer Flanke des $\overline{\text{DONE}}$-Impulses reagieren kann.

Um den $\overline{\text{DONE}}$-Impuls der FPU erkennen zu können, benutzt man die in *Bild 8* gezeigte Schaltung. Jeder $\overline{\text{SPC}}$-Impuls, der auftritt, während das Transfer-Select-Signal ($\overline{\text{XFER}}$) inaktiv ist, wird als $\overline{\text{DONE}}$-Impuls interpretiert, der dann in einem Flipflop zwischengespeichert wird. Wenn die CPU 68000 einen Lesezyklus von einer Adresse ausführt, die das $\overline{\text{POLL}}$-Select-Signal erzeugt, wird der Flipflop-Inhalt auf das höchstwertige Bit des Datenbus gegeben (Leitung D15). Damit ist es dem 68000 möglich, auf das FPU-Signal zu warten, indem eine kurze MOVEW- und BPL-Schleife eingelegt wird.

Wie aus Bild 7 hervorgeht, erzeugt das mit SPC bezeichnete Flipflop die Flanken für den $\overline{\text{SPC}}$-Impuls (aufgrund des Signals $\overline{\text{SPCT}}$). Die Timing-Kette TA, TB stellt die Freigabesteuerung für den Puffer zur Verfügung, der den $\overline{\text{SPC}}$-Anschluß an der FPU ansteuert. Entsprechend der Richtung des Transfers wird entweder TB oder TC benutzt, um den $\overline{\text{SPC}}$-Impuls abzuschließen. Es ist zu beachten, daß die Timing-Kette einen Speicherzyklus mit voller Geschwindigkeit von vier Taktperioden beim Zugriff auf die FPU vorschreibt. Anderenfalls kann die Schaltung kein Data-Acknowledge-Signal für die 68000-CPU erzeugen (DTACK, nicht gezeigt).

Das Flipflop im $\overline{\text{DONE}}$-Pulsdetektor (Bild 8) wird gelöscht, wenn die Daten in die FPU geschrieben werden, und gesetzt, wenn die FPU anzeigt, daß ihre Operationen beendet sind, indem sie über den $\overline{\text{SPC}}$-Anschluß einen „Low"-Impuls ausgibt.

Der Zweck des Hardware-Interface ist es, das Verhalten des $\overline{\text{SPC}}$-Anschlusses beim 32000 zu simulieren. Der Rest der Interface-Task besteht aus dem Präsentieren von Kommandos und Operanden für die FPU mit Hilfe von Dummy-Adressen, die die Leitungen ST0 und ST1 simulieren. Weil verschiedene Adreßumsetzungen vom 32000 und 68000 benutzt werden, gibt es zwei spezielle Punkte zu beachten. Erstens wird ein Byte mit gerader Adresse vom 68000 in die höherwertige Hälfte des Datenbusses transferiert. Die FPU erwartet allerdings diese Information auf der niedrigwertigen Hälfte. Das bedeutet, wenn ein einzelnes Byte zu oder von der FPU transferiert wird, muß die FPU entweder eine ungerade Adresse haben oder als niedrigwertige Hälfte eines 16-Bit-Wortes mit einer geraden Adresse transferiert werden.

Zweitens transferiert die CPU 68000 32-Bit-Operanden als zwei aufeinanderfolgende 16-Bit-Worte, wobei das höherwertige Wort zuerst kommt. Die FPU folgt der Vereinbarung für den 32000 und erwartet den Empfang des niederwertigen Wortes als erstes. Die SWAP-Instruktion des 68000 läßt sich zum Umkehren der beiden Hälften eines Doppelwortes vor der Übertragung heranziehen.

Zum Schluß zeigt *Tabelle 3* ein Beispiel für das Listing der Ausführung einer ADDF-Instruktion. Es wird

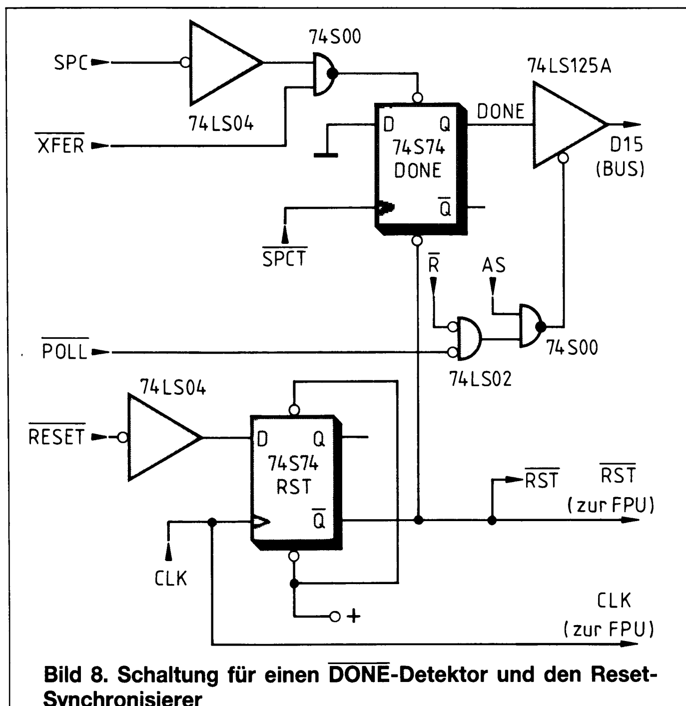


Bild 8. Schaltung für einen $\overline{\text{DONE}}$-Detektor und den Reset-Synchronisierer

vorausgesetzt, daß die Adresse in Form von 06XXXX zur Erzeugung des $\overline{\text{XFER}}$-Signals benutzt wird, während die Adresse in Form der 07XXXX zur Erzeugung des $\overline{\text{POLL}}$-Signals herangezogen wird. Die Adreßregister A1...A3 werden so konfiguriert, daß die Adreßleitungen A4 und A5 entsprechende Kombinationen von ST0 und ST1 zur Verfügung stellen.

Tabelle 3. Beispiel für eine Fließkomma-Operation (Addition einfacher Genauigkeit)

```
*       Register contents
*
*       A0=00070000     address of DONE flip-flop
*       A1=00060010     address for ST=01 (transfer operand)
*       A2=00060020     address for ST=10 (read status word)
*       A3=00060030     address for ST=11 (broadcast ID byte)
*
*       D0=000000BE     ID byte for ADDF instruction
*       D1=00000184     operation word for ADDF (bytes swapped)
*       D2=3F800000     first operand (=1.0)
*       D3=3F800000     second operand (=1.0)
*       D4              receives status word from FPU
*       D5              receives result from FPU
*       D7              scratch register for DONE bit test

START   MOVE.W D0, (A3)send ID byte
        MOVE.W D1,( A1)send operation word
        SWAP D2         send operands. The swapping is included
        MOVE.L D2, (A1)                because the FPU expects the
        SWAP D2                        least significant word first.
        SWAP D3                        It can be avoided with care.
        MOVE.L D3, (A1)
        SWAP D3
*
POLL    MOVE.W (A0), D7check DONE flip-flop & loop until FPU
        BPL POLL        has finished
*
*
*
        MOVE.W (A2), D4read status word from FPU
        MOVE.L (A1), D5 read result
        SWAP D5         swap halves of result
```

Werner Trattnig

Virtuelle Speicherverwaltung in µC-Systemen

Die erste Generation von Computersystemen auf der Basis von Mikroprozessoren war mit 8-Bit-CPUs aufgebaut, z. B. den Typen 8080, 6800 oder 6502, die alle mit 16-Bit-Adressen arbeiten, so daß sich ein Adreßbereich von 64 KByte ergibt. Mit fallenden Preisen für dynamische RAMs wurde es wirtschaftlicher, alle Adreßbereiche mit Schreib-Lese-Speichern zu besetzen, die nicht von ROM- oder E/A-Bausteinen besetzt waren. Üblich ist in diesen Systemen daher eine direkte Korrespondenz zwischen logischen Adressen, das sind die Adressen, die der Mikroprozessor direkt über seinen Adreßbus ausgibt, und den physikalischen Adressen, das sind die tatsächlichen Speicherplätze, in denen Daten gespeichert sind. Bald machten sich allerdings die Grenzen des Adreßbereiches von 64 KByte bemerkbar, insbesondere bei Anwendungen mit einem Platten-Betriebssystem oder hochauflösender Grafik. Man entwickelte daher verschiedene Techniken zur Verwaltung größerer physikalischer Adreßbereiche, die sich mit einem logischen Adreßumfang von 64 KByte beherrschen lassen. Häufig findet man die Aufteilung des physikalischen Speichers in Bänke, wobei jede Bank den gleichen logischen Adreßbereich überdeckt. Mit Hilfe softwareaktivierter Schalter wird der Zugriff auf einzelne Bänke gesteuert. Ein Nachteil ist hier allerdings, daß die Speicherverwaltung nur mit großem Software-Aufwand abläuft. Es waren daher Techniken notwendig, die große Speicherkapazitäten verwalten und dabei die Zugriffszeit nicht unzulässig erhöhen.

Bei 16-Bit-Mikroprozessoren, die ältere 8-Bit-Typen ablösten, z. B. die Bausteine 8086 und 68000, versuchte man, das Adressierproblem zu lösen, indem der Adreßbus erweitert wurde. So kann man beispielsweise mit einem 32-Bit-Adreßbus einen logischen Adreßbereich von 4 GByte abdecken. Allerdings bedeutet dies nicht, daß eine Speicherverwaltung überflüssig ist. Obwohl die Preise für Speicherbausteine kontinuierlich fallen, bleiben die Kosten für 1 Bit in einem Halbleiterspeicher immer noch wesentlich über denjenigen eines Massenspeichersystems, z. B. Festplatten-Laufwerken. Kosten, physikalische Abmessungen und Leistungsaufnahme von Halbleiterspeichern werden auch in der überschaubaren Zukunft noch so groß bleiben, daß nur ein kleiner Teil des logischen Adreßbereiches als physikalisches RAM gebaut wird. Computerhersteller verkaufen heute beispielsweise ihre Geräte in der Grundausstattung mit 1 MByte RAM, wobei die Option für zusätzliche Schreib-Lese-Speicher offenbleibt.

In der Praxis ist es nicht immer wirtschaftlich, zusätzliche RAM-Kapazität in ein System einzubauen. Auch die Kosten für Festplattensysteme fallen, so daß es wirtschaftlich durchaus sinnvoll ist, einen Mikrocomputer mit einem Massenspeicher von 20 MByte auszustatten. Es ist nämlich kosteneffektiver, große Programmodule oder Datenblöcke auf Platte abzulegen und diese nur in das RAM zu laden, wenn sie benötigt werden. Im einfachsten Falle speichert man bestimmte Programm- oder Datenblöcke auf bekannten Platten-Sektoren und transferiert diese in vorher definierte RAM-Bereiche, um mit ihnen zu arbeiten. Für bestimmte Anwendungen im Ein-Benutzer-Betrieb ist diese „Overlay"-Technik sehr preiswert und wirtschaftlich.

Overlay-Technik nur für einfache Systeme

In der Mehrzahl aller Fälle ist diese einfache Technik allerdings nicht brauchbar. Beispielsweise können in einem Mehr-Benutzer-System leicht Konflikte auftreten, wenn zwei Benutzer verschiedene Overlays im gleichen RAM-Block unterbringen möchten. Eine wesentlich elegantere und wirkungsvollere Lösung ist die Verwendung eines virtuellen Speicherkonzeptes. Obwohl es virtuelle Speicher in Großrechnern und größeren Minicomputern schon seit einigen Jahren gibt, ist es erst seit

kurzer Zeit möglich, diese kosteneffektiv auch in Mikrocomputern zu implementieren.

In einem virtuellen Speichersystem werden die logischen Adressen, die der Mikroprozessor ausgibt (diese tragen auch die Bezeichnung „virtuelle Adressen") in einem Sekundärspeicher abgebildet, z. B. einer Festplatten-Einheit, so daß der Benutzer den Eindruck hat, daß alle logischen Adressen tatsächlich auch implementiert sind. Ein System, in dem jeder Speicherzugriff auch ein Zugriff auf die Festplatte darstellt, würde allerdings sehr langsam arbeiten, was nicht akzeptabel ist. Deshalb sind einige der Sekundär-Speicherplätze wiederum im RAM abgebildet. *Bild 1* zeigt dieses Grundkonzept.

Eine typische Festplatteneinheit mit 16 MByte ist beispielsweise in 4096 Spuren mit jeweils acht Sektoren zu 512 Byte konfiguriert. Auf diese Weise läßt sich jedes Byte auf der Festplatte mit Hilfe seiner Spurnummer (12 Bit), der Sektornummer (3 Bit) und dem Offset innerhalb des Sektors (9 Bit) spezifizieren, so daß sich eine 24-Bit-Festplattenadresse ergibt. Um die Sache zu vereinfachen, kann man voraussetzen, daß die virtuellen 24-Bit-Adressen, die der Mikroprozessor ausgibt, direkt auf den 24-Bit-Festplattenadressen abgebildet sind, obwohl dies nicht unbedingt notwendig ist, solange dem Betriebssystem die tatsächliche Korrespondenz bekannt ist.

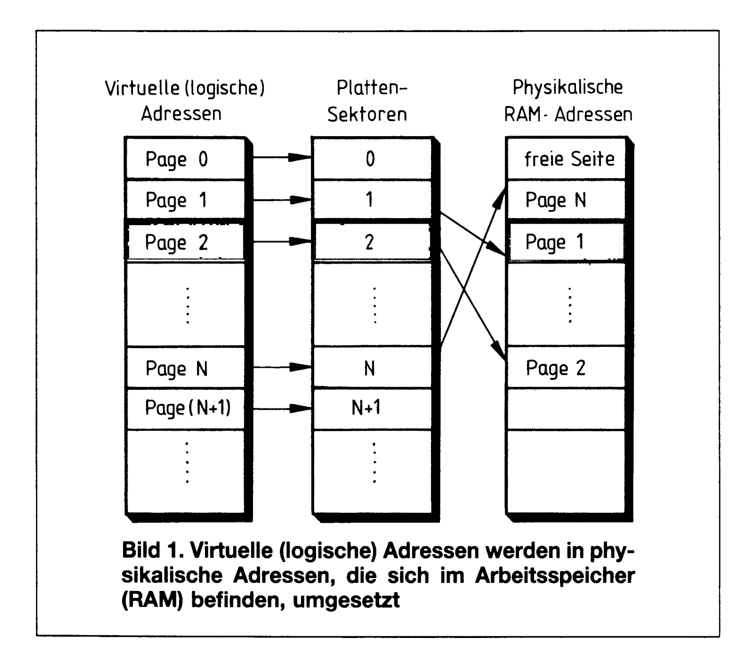


Bild 1. Virtuelle (logische) Adressen werden in physikalische Adressen, die sich im Arbeitsspeicher (RAM) befinden, umgesetzt

Die Beziehung zwischen virtuellen Adressen und Platten-Speicherplätzen steuert der Programmierer. Die Beziehung zwischen virtuellen Adressen und physikalischen Adressen steuert dagegen das Betriebssystem sowie eine besondere Speicher-Verwaltungshardware. Die Aufgabe des Speicher-Verwaltungssystems ist die Umsetzung der virtuellen Adresse, die der Mikroprozessor ausgibt, in die tatsächliche physikalische Adresse, unter der die betreffenden Daten zu finden sind. Wenn es keine physikalische Adresse gibt, weil die erforderlichen Daten noch nicht ins RAM gebracht worden sind, müssen ein oder mehrere Plattensektoren einschließlich des Sektors mit der entsprechenden virtuellen Adresse in einen passenden RAM-Bereich geladen werden, so daß eine physikalische Adresse vorhanden ist.

Ein virtueller Speicher ist in Mehr-Benutzer-Systemen von Vorteil. Hier müssen separate Prozesse gleichzeitig ausgeführt werden, wobei das Betriebssystem im Zeit-Multiplex-Verfahren die System-Ressourcen zwischen ihnen umschaltet. Ein virtueller Speicher ermöglicht es jedem Benutzer, einen vollständigen virtuellen Adreßbereich zu benutzen, unabhängig davon, wieviel tatsächliche RAM-Kapazität vorhanden ist. Die Vergrößerung der RAM-Kapazität erhöht in der Regel den Systemdurchsatz, weil weniger Zeit vergeht, in der die Daten zwischen Primär- und Sekundär-Speicher transferiert werden müssen.

Darüber hinaus verfügt das Betriebssystem über einen vollständigen virtuellen Speicherbereich, so daß es so groß wie erforderlich gemacht werden kann. In Systemen, bei denen das Betriebssystem sich den physikalischen Adreßbereich mit dem Anwendungsprogramm teilen muß, ist es kaum möglich, das Betriebssystem zu verbessern, ohne gleichzeitig die Speicherkapazität zu erhöhen.

Virtuelle Speicherverwaltung löst alle Probleme

Obwohl das Konzept der virtuellen Speicherverwaltung recht einfach ist, hängt der Aufwand bei der Implementierung vom Mikroprozessortyp ab. Das primäre Ziel ist, daß der Mechanismus des virtuellen Speichers für den Benutzer vollständig transparent ist, was bedeutet, daß die Speicherverwaltung im Betriebssystem eingebettet sein muß. Viele Untersuchungen haben gezeigt, daß bei den meisten Anwendungen die CPU mehr Zeit dazu verwendet, Befehle und Daten aus dem Speicher zu holen, als für die eigentliche Verarbeitung der Daten notwendig ist. Damit ist klar, daß die Geschwindigkeit, mit der die logischen Adressen in physikalische Adressen übersetzt werden, den gesamten Systemdurchsatz beeinflußt.

Immer, wenn die CPU eine virtuelle Adresse ausgibt, liegt eine von zwei Bedingungen vor: Die erforderlichen Daten können bereits im RAM sein, wobei die tatsächlichen physikalischen Adressen innerhalb kürzester Zeit in die entsprechenden virtuellen Adressen zu übersetzen sind. Wenn dies nicht der Fall ist, müssen die Daten zunächst vom Sekundärspeicher geholt werden. Dies bedeutet in der Praxis, daß ein oder mehrere Plattensektoren in das RAM geladen werden und in vielen Fällen, daß der vorhergehende RAM-Inhalt dafür zunächst auf die Platte zurückgebracht werden muß. Das Betriebssystem muß mit einer Strategie ausgestattet werden, die festlegt, welche RAM-Seiten ausgelagert werden können, um Platz für die zu überspielenden Plattensektoren zu schaffen. Es gibt zwei prinzipielle Verfahren dafür: Segmentierung und Demand-Paging.

Segmentierung

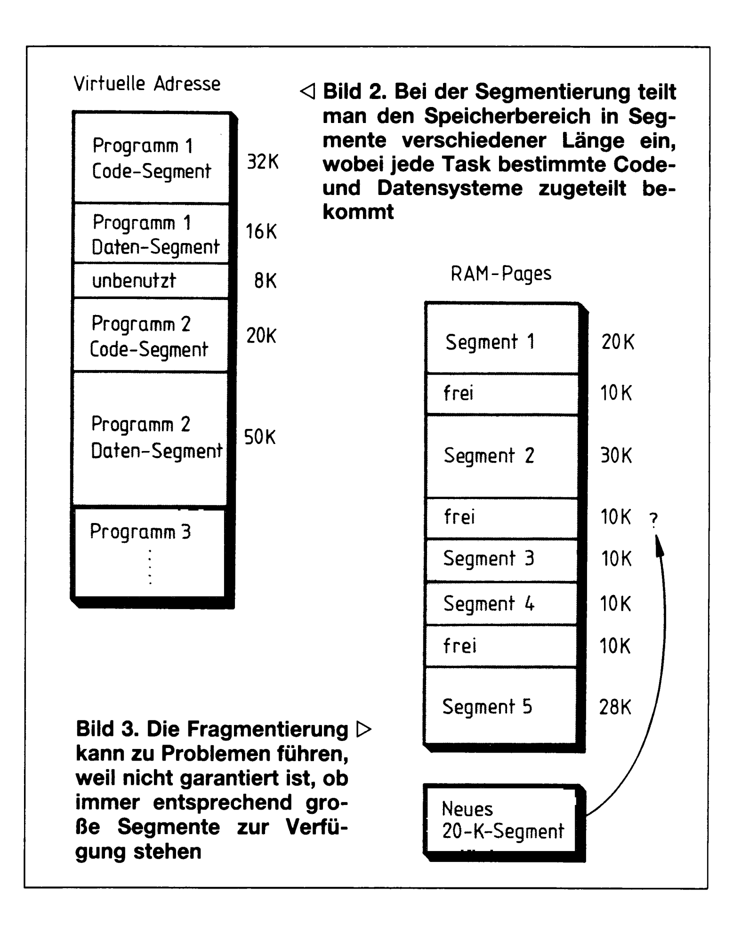
Das Segment-Verfahren wird von Minicomputer-Herstellern seit Ende der 50er Jahre bereits verwendet und ist seit einigen Jahren bei Mikrocomputern, wie z. B. Systemen mit dem 8086, zu finden. Hierbei ist der logische Adreßbereich in Segmente verschiedener Länge eingeteilt (*Bild 2*), und jeder Task ist ein oder mehrere Segmente für den Programm-Code und die Daten zugeordnet. Beim 8086 sind beispielsweise für jede Task vier Segmente definiert: Code-Segment, Stack-Segment, Daten-Segment und ein alternatives Daten-Segment.

Der wichtigste Vorteil der Segmentierung ist die Tatsache, daß es mit diesem Verfahren möglich ist, einen virtuellen Speicher mit wenig Hardware zu implementieren. Weil jede Task mit einer kleinen Zahl von Segment-Deskriptoren definiert werden kann, lassen sich die Definitionen für verschiedene Tasks in Hardware-Registern oder unter RAM-Adressen ablegen, so daß sich jedes Segment auch mit einer Schutzebene versehen läßt. Z. B. enthält die Speicherverwaltungseinheit MC68451 insgesamt 32 Segment-Deskriptor-Register, wodurch es dem Betriebssystem möglich ist, zwischen verschiedenen Tasks umzuschalten. Weil dies einfach in Form von Silizium zu implementieren ist und die Funktion bei relativ einfachen Anwendungen, bei denen die Anzahl der Tasks, die gleichzeitig auszuführen sind, sehr klein ist, sehr gut arbeitet, wurde die Segmentierung als Speicherverwaltungs-Strategie bei den meisten modernen 16-Bit- und 32-Bit-Mikroprozessoren übernommen.

Allerdings enthalten alle segmentierten Architekturen einen grundsätzlichen Nachteil, der die Konzipierung eines Multi-Tasking-Betriebssystems sehr kompliziert macht. Die Ursache für die Schwierigkeit liegt darin, daß entweder das gesamte Segment im RAM vorhanden ist oder nicht; es ist nicht möglich, nur Teile eines Segmentes zu übertragen. Daraus ergibt sich als erstes, daß die maximale Segmentgröße durch die im System zur Verfügung stehende RAM-Kapazität begrenzt ist. Wenn beispielsweise ein bestimmtes Programm ein 70-KByte-Code-Segment benötigt, ist es nicht möglich, dieses auf einem System mit nur 64 KByte RAM laufen zu lassen, weil das Betriebssystem dieses Code-Segment überhaupt nicht laden kann. Aus dem gleichen Grund kann ein Programm mit einem 40-KByte-Code-Segment und einem 32-KByte-Daten-Segment nicht auf einem 64-KByte-System laufen.

Eine wesentlich größere Schwierigkeit ergibt sich aus der Fragmentierung, die entsteht, wenn die Segmentgrößen eine Potenz von 2 sind, was häufig der Fall ist. Ein 40-KByte-Block von Code muß dann in einem 64-KByte-Segment untergebracht werden, so daß 24 KByte Speicherkapazität verschwendet sind.

Externe Fragmentierung kann noch größere Probleme hervorrufen. *Bild 3* zeigt einen hypothetischen Fall, bei dem 30 KByte freie RAM-Kapazität vorhanden sind, aber kein Platz für ein 20-KByte-Segment ist. Es gibt verschiedene Möglichkeiten, wie das Betriebssystem eine solche Situation meistern kann, die bei segmentierten Systemen sehr häufig auftreten. Eine Methode ist z. B., alle existierenden Segmente in Richtung auf das Ende des RAM-Adreßbereiches zu verschieben, um die unbenutzten Speicherteile für das neue Segment freizumachen. In der Praxis gibt es einige Architekturen, die die dynamische Relocation von Code und Daten erlauben, so daß diese Methode auch benutzt werden kann.



◁ **Bild 2. Bei der Segmentierung teilt man den Speicherbereich in Segmente verschiedener Länge ein, wobei jede Task bestimmte Code- und Datensysteme zugeteilt bekommt**

Bild 3. Die Fragmentierung ▷ kann zu Problemen führen, weil nicht garantiert ist, ob immer entsprechend große Segmente zur Verfügung stehen

Eine andere Möglichkeit ist es, daß das Betriebssystem ganz einfach wartet, bis ein anderer Prozeß abgeschlossen ist, wobei man darauf hofft, daß genügend freier RAM-Bereich entsteht, daß das wartende Segment geladen werden kann. Während dieses Verfahren einfach zu implementieren ist, kann es zu langen Verzögerungen führen, die den wirklichen Zweck dieses virtuellen Speicherverfahrens sinnlos machen.

Das für die meisten Systeme übernommene Verfahren ist, ein Segment passender Größe auszulagern, um Platz für ein neues Segment zu schaffen. Falls es kein Segment von exakt gleicher Größe gibt, muß ein größeres Segment ausgelagert werden, so daß wiederum wertvoller RAM-Bereich verschwendet wird. Im ungünstigsten Fall wird das ausgelagerte Segment kurz darauf wieder benötigt, was zu Folge hat, daß beispielsweise zwei oder mehrere kleinere Segmente entfernt werden müssen, um genügend Raum zu schaffen. Weil es nicht selten vorkommt, daß Segmente 64 KByte oder mehr umfassen, kann die Zeit für die Massentransfers zwischen RAM und Platte die Gesamtleistung des Systems merklich beeinträchtigen.

Demand-Paging

Seit mehr als zehn Jahren benutzen Großrechner- und Minicomputer-Hersteller eine wesentlich effizientere Speicherverwaltungsmethode, die die Bezeichnung „virtueller Speicher nach dem Demand-Paged-Verfahren“ (Demand Paged Virtual Memory) hat. Bei dieser Methode wird der logische Adreßbereich des Speichers in eine große Zahl kleiner Einheiten eingeteilt, die die Bezeichnung „Pages“ (Seiten) tragen. Eine typische Seitengröße beträgt 512 Byte. Der physikalische Speicherbereich ist in ähnlicher Weise in Einheiten gleicher Größe aufgeteilt, die die Bezeichnung „Page Frames“ tragen. Jede logische Seite kann zu jedem Page Frame transferiert werden. Dies bedeutet, daß zwei aufeinanderfolgende Adressen in einer Seite auch im Page Frame hintereinander angeordnet sind, allerdings werden zwei aufeinanderfolgende logische Seiten nicht unbedingt in aufeinanderfolgenden Page-Frames abgebildet.

Diese einfache Paging-Strategie überwindet alle Nachteile der Segmentierung. Weil die Grundeinheit für das Holen oder Auslagern eine Seite ist, gibt es keine Notwendigkeit, das gesamte Segment in dem Massenspeicher zu bringen, um Platz für eine neue Seite zu schaffen. Das Betriebssystem ist nicht gezwungen, ein Segment einer bestimmten Größe zu finden, sondern soll statt dessen die Frame-Page auslagern, die mit geringster Wahrscheinlichkeit in der nächsten Zukunft benötigt wird. Es gibt in der Praxis immer noch eine gewisse interne Fragmentierung, weil eine vollständige 512-Byte-Seite ausgelagert werden muß, auch wenn nur ein Teil davon wirklich benötigt wird, allerdings ist der Umfang des verschwendeten Speicherbereiches merklich reduziert.

Bild 4. Der virtuelle Speicherzugriff nach dem Demand-Paged-Verfahren bei der 32000-Familie benutzt eine Zwei-Ebenen-Tabelle, um die erforderliche RAM-Kapazität zu verringern

Das Demand-Paged-Verfahren hat zwei Eigenschaften, die es gegenüber jeglicher Segmentierung überlegen machen: Erstens wird die Zeit, die das Betriebssystem zum Verschieben der Datenblöcke zwischen RAM und Platte benötigt, merklich reduziert, so daß sich ein höherer Systemdurchsatz ergibt. Zweitens ist es nicht notwendig, daß ein Prozeß seinen gesamten Code- oder Daten-Bereich im RAM haben muß, weil jede nicht vorhandene Seite sehr schnell transferiert werden kann, wenn dies erforderlich ist.

Dies bedeutet, daß die Anzahl der Prozesse, die gleichzeitig ablaufen können, normalerweise nicht von der Größe des RAMs abhängt; Software, die beispielsweise auf einem System mit 1 MByte RAM entwickelt wurde, kann unverändert auch auf einem System laufen, das nur mit 128 KByte RAM ausgestattet ist. Der einzige Unterschied ist eine mögliche Leistungsverschlechterung, weil mehr Page-Transfers notwendig werden. Dies ist ein wesentlicher Unterschied zu segmentierten Verfahren, bei denen die Größe des RAMs die Anzahl der gleichzeitig ablaufenden Prozesse begrenzt.

Obwohl das Demand-Paging schon seit einiger Zeit erfolgreich in Großrechnern und Minicomputern Anwendung findet, kann man es noch nicht solange in Zusammenhang mit Mikrocomputern verwenden, weil eigentlich alle Mikroprozessorfamilien, die für die Unterstützung virtueller Speicherverfahren konzipiert sind, unter dem Gesichtspunkt der Segmentierung entwickelt wurden. Eine effizient arbeitende Implementierung des virtuellen Speichers nach dem Demand-Paged-Verfahren läßt sich nur dann erreichen, wenn die notwendigen Mechanismen bereits von Anfang an in das Mikroprozessorkonzept einbezogen wurden. Bisher bietet nur die 32000-Familie die entsprechenden Merkmale.

Das Entwicklungsziel der 32000-Familie war eine Ein-Chip-VAX, deshalb ist die virtuelle Speicherverwaltung nach dem Demand-Paged-Verfahren vollständig in der Architektur integriert. Die heutigen Möglichkeiten der Herstellung reichen allerdings noch nicht aus, alles auf einem Chip unterzubringen, so daß die meisten Speicherverwaltungsfunktionen von einem separaten Coprozessor, der Speicherverwaltungseinheit NS32082, ausgeführt werden. Allerdings bieten die meisten CPUs der 32000-Familie spezielle Merkmale, die in die Architek-

tur eingebaut sind, z. B. den Befehlsabbruch, so daß CPU und MMU als ein System zusammenarbeiten können.

Bei 32000-Systemen ist der logische Adreßbereich von 16 MByte in 32 768 Seiten zu jeweils 512 Byte aufgeteilt. Dies ist eine sehr praktische Seitengröße, bei der diese Kapazität bei den üblichen Festplatten auch der Sektorgröße entspricht. Physikalischer Frame, die logische Seite und der Plattensektor haben daher gleiche Größe. Die 24-Bit-Adresse läßt sich in zwei Felder teilen: Die oberen 15 Bits bestimmen die Seite, in der der adressierte Speicherplatz zu finden ist. Die MMU ersetzt diese durch die 15 höherwertigen Bits der wirklichen RAM-Adressen, wenn die Seite sich derzeit im RAM befindet. Die unteren 9 Bits definieren die Position des adressierten Bytes innerhalb der Seite. Dies verändert sich nicht, denn die Seitengröße entspricht der Frame-Größe.

Um die Umsetzung von logischen auf physikalische Adressen durchführen zu können, benötigt das System eine Umsetztabelle, die, falls diese existieren, die physikalischen Adressen für die 32 768 logischen Seiten enthält. Ein Teil der Mitglieder der 32000-Familie arbeitet mit einer Adreßbreite von 24 Bit. Man geht davon aus, daß die acht höchstwertigen Adreßbits auf Null liegen. Bei 32 Bit Adreßbreite ergibt sich ein Adressierbereich von 4 GByte. Jeder Eintrag in die Umsetztabelle nimmt deshalb 4 Byte ein, so daß sich insgesamt 131 072 Bytes ergeben. Programme, die nicht den gesamten Adreßbereich von 16 MByte benutzen, benötigen auch nicht die vollständige Umsetztabelle. Allerdings belegt eine Ein-Ebenen-Umsetztabelle eine nicht akzeptierbare große Anzahl von RAM-Speicherplätzen.

Um die zum Speichern der Umsetztabellen erforderliche RAM-Kapazität zu reduzieren, benutzt die 32000-Familie eine Zwei-Ebenen-Tabelle, wie sie in *Bild 4* dargestellt ist. Die 32 768 Tabelleneintragungen sind in 256 Untertabellen aufgeteilt, die als „Pointer-Tabellen" bekannt sind. Jede Pointer-Tabelle umfaßt eine gesamte Seite und enthält 128 Eintragungen, wobei jede Eintragung ein 32-Bit-Wort ist, das die tatsächliche physikalische Adresse der Seite enthält, wenn sie sich im RAM befindet, sowie verschiedene Status-Bits, die im folgenden beschrieben werden. Weil Programme üblicherweise lokal ausgeführt werden, ist es selten notwendig, mehr als eine oder zwei Pointer-Tabellen gleichzeitig im RAM zu haben, so daß sich diese normalerweise auf der Platte befinden und nur geladen werden, wenn dies erforderlich ist.

Obwohl die 128 Eintragungen einer Pointer-Tabelle 128 aufeinanderfolgende virtuelle Adressen beschreiben, müssen die Pointer-Tabellen nicht kontinuierlich oder in einer bestimmten Reihenfolge angeordnet sein. Statt dessen enthält eine weitere Tabelle die virtuellen Anfangsadressen jeder Pointer-Tabelle. Diese Tabelle hat eine Länge von 1024 Byte und trägt die Bezeichnung „Page-Tabelle". Sie muß sich im RAM befinden. Weil es für jeden laufenden Prozeß eine separate Page-Tabelle gibt, muß man der MMU die virtuelle Adresse für den

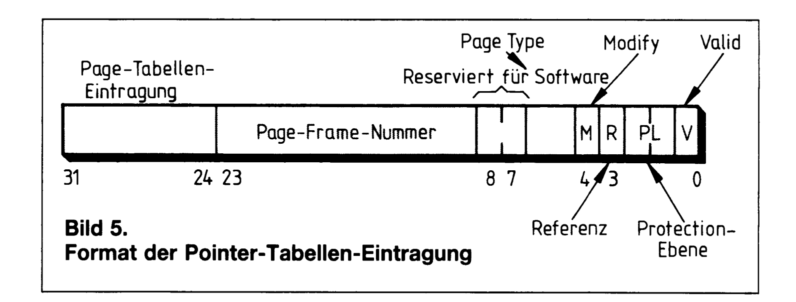


Bild 5.
Format der Pointer-Tabellen-Eintragung

Anfang der entsprechenden Page-Tabelle geben. Die MMU legt diese in einem der zwei internen Register mit der Bezeichnung „Page-Table-Base-Register" ab.

Mit der Zwei-Ebenen-Seitentabelle werden die Adressen in drei Felder zurückverwandelt, wie das Bild 4 zeigt. Die acht höchstwertigen Bits dienen zur Auswahl von einem der 256 Einträgen in der laufenden Seitentabelle. Die Daten, die aus der Seitentabelle gelesen werden, stellen die virtuelle Startadresse einer der Pointer-Tabellen dar. Die nächsten sieben höchstwertigen Bits der Originaladresse wählen eine der 128 Einträge der betreffenden Pointer-Tabelle aus, die die unteren 15 Bit der erforderlichen physikalischen Adressen enthält (sowie weitere verschiedene Status-Bits). Die neun niedrigwertigen Bits der Originaladressen werden angehängt, um die endgültigen physikalischen 24-Bit-Adressen zu formen.

Ganz offensichtlich verlangsamen die beiden Speicherzugriffe zur Umsetzung der logischen Adresse in physikalische Adressen den Ablauf, insbesondere dann, wenn man sich daran erinnert, daß mehr als die Hälfte aller Befehle in einem typischen Programm externe Speicherzugriffe sind und keine internen Datenoperationen. Obwohl jede Art virtueller Speicher für seine Verwaltung Zeit braucht, ist dieser Nachteil beim 32000-System durch den Hochgeschwindigkeits-Cache-Speicher kompensiert, der sich in der MMU befindet. Dieser Cache-Speicher ist inhaltadressierbar und enthält die 32 Seiten, auf die am häufigsten zugegriffen wurde; sowie deren umgesetzte physikalische Adressen. Wenn die Adresse, die von der CPU ausgegeben wird, einer Cache-Eintragung entspricht und die Schutzebene der Seite den Zugriff erlaubt, wird die physikalische Adresse innerhalb von 100 ns ausgegeben (bei 10-MHz-Takt). Wenn die Adresse sich nicht im Cache befindet, aber die Pointer-Tabelle im RAM ist, wird die physikalische Adresse nach 2 µs verfügbar. Wenn die Pointer-Tabelle in das RAM umgeladen werden muß, ist die Verzögerung etwas länger.

Auch hier führt die Tatsache, daß die meisten Programme lokal ablaufen, dazu, daß die Leistung wesentlich höher ist, als es auf den ersten Blick erscheint. Messungen haben gezeigt, daß der Cache die erforderliche Seite in 97...98 % der Fälle enthält und daß die gewichtete mittlere Verzögerung vor dem Verfügbarwerden der physikalischen Adresse lediglich 157 ns beträgt.

Bild 5 zeigt das Format einer Pointer-Tabellen-Eintragung. Die fünf niedrigwertigen Bits enthalten die Status-

informationen der entsprechenden logischen Seite. Bits 9...23 spezifizieren die physikalische Startadresse. Bits 24...30 sind reserviert für zukünftige Erweiterungen und das höchstwertige Bit benutzt man als Bank-Auswahlbit. Mit dessen Hilfe kann man das RAM in zwei Bänke einteilen, so daß eine vollständige virtuelle Maschine implementiert werden kann.

Das V-Bit („Valid Status") zeigt an, ob ein Eintrag für die Adreßumsetzung benutzt werden kann. Wenn ein Frame ausgelagert wurde, um für eine ankommende Seite Platz zu schaffen, wird die physikalische Adresse nicht geändert, aber das V-Bit zurückgesetzt, um anzuzeigen, daß der Eintrag nicht mehr gültig ist. Die beiden PL-Bits spezifizieren die Schutzebene der Seite. In Benutzer-Betriebsart sind drei Zugriffsbedingungen möglich: kein Zugriff, nur Lesen und Schreiben/Lesen. In der Supervisor-Betriebsart gibt es die Zugriffsbedingungen nur Lesen und Schreiben/Lesen.

Die beiden übrigen Statusbits sind das R-Bit („Referenced"), das automatisch dann gesetzt ist, wenn in der zugegriffenen Seite gelesen oder geschrieben wird, und das M-Bit („Modified"), das gesetzt wird, wenn die Daten in die Seite geschrieben werden, während sich diese im RAM befindet. Der wichtigste Zweck dieser Status-Bits ist es, dem Entwickler des Betriebssystemes die Implementierung eines effizient arbeitenden Page-Swapping-Algorithmus zu erleichtern.

Weil im allgemeinen keine freie Page-Frame vorhanden ist, wenn eine neue Seite in das RAM gebracht wird, muß das Betriebssystem eine Page-Frame auswählen, die zu überschreiben ist. Durch periodisches Lesen und Rücksetzen des R-Statusbits in allen Seiten- und Pointer-Tabellen kann das Betriebssystem die Häufigkeit der Seitenzugriffe überwachen und die am wenigsten benutzte Seite aussondern. Untersuchungen haben gezeigt, daß mit hoher Wahrscheinlichkeit diese Seite in der überschaubaren Zukunft nicht wieder erforderlich ist. Das M-Statusbit sagt dem Betriebssystem, ob der entfernte Page-Frame der erste ist, der zur logischen Seite zurückgebracht wird, woher er kam.

MMU-Hardware

Weil die Speicherverwaltungseinheit NS32082 ein Slave-Prozessor ist, der vollständig in die CPU-Architektur integriert wurde, ist der Aufwand für die Implementierung eines virtuellen Speichers nach dem Demand-Paged-Verfahren minimal. *Bild 6* zeigt eine Blockschaltung der MMU, während *Bild 7* die einfache Verbindung zu CPU und Systembus erkennen läßt.

Die MMU enthält zehn Register, auf die der Benutzer zugreifen kann und die in zwei Gruppen aufgeteilt sind. Sechs der Register bieten Debugging-Unterstützung, so daß der Benutzer Breakpoints setzen oder den Programmfluß verfolgen kann, während drei Register für virtuelle Speicherfunktionen bestimmt sind. Das Statusregister MSR dient sowohl dem Debugging als auch für virtuelle Speicherfunktionen.

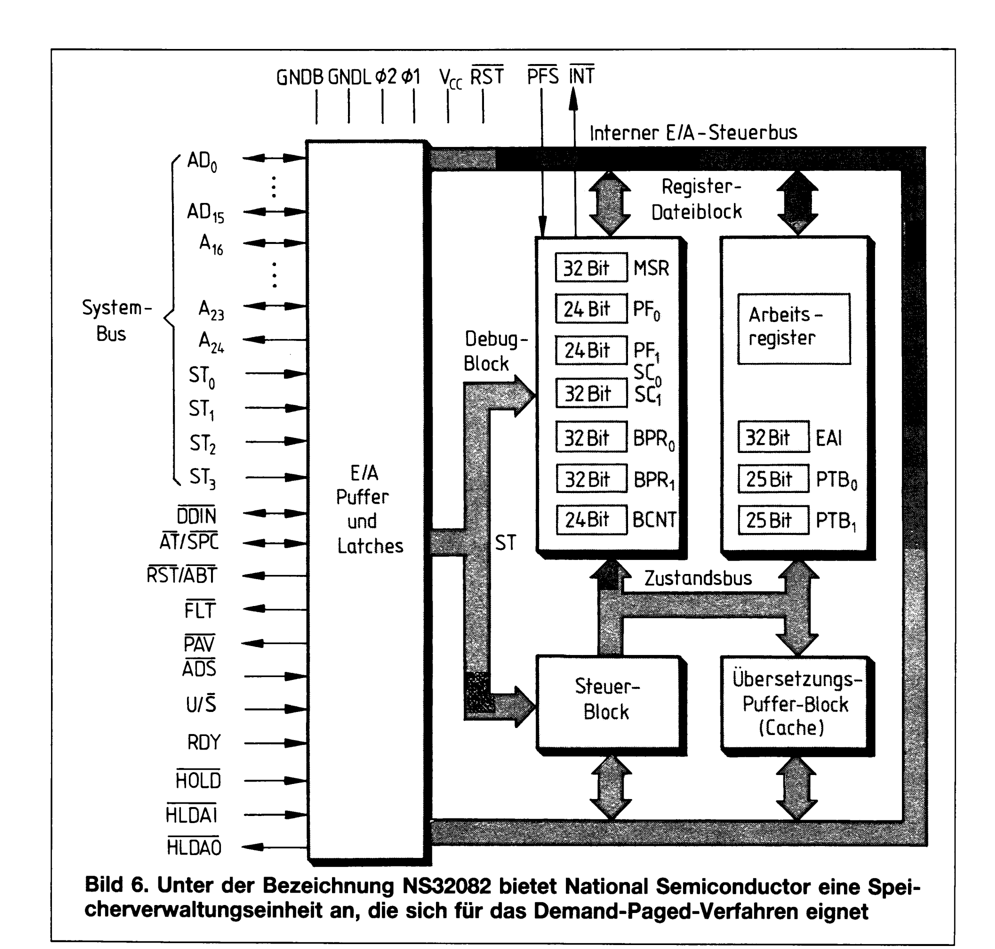


Bild 6. Unter der Bezeichnung NS32082 bietet National Semiconductor eine Speicherverwaltungseinheit an, die sich für das Demand-Paged-Verfahren eignet

Zum Zwecke der dynamischen Adreßumsetzung sind die interessanten Register das Page-Table-Base-Register PTB0 und PTB1 sowie das Error/Invalidate-Adreß-Register EIA. Die PTB-Register spezifizieren die Basisadresse der Seitentabelle auf Ebene 1. Sie können jederzeit gelesen oder modifiziert werden, indem die MSR- oder LMR-Befehle der CPU ausgeführt werden. Sie werden üblicherweise beim Anfang einer Task geladen. Normalerweise liest die MMU die Basisadresse von PTB1, während die CPU in Benutzer-Betriebsart das Programm arbeitet und von PTB0, wenn sie in Supervisor-Betriebsart arbeitet. Hierfür wird das Dual-Space-Bit im MSR gelöscht, wobei allerdings der Benutzer die MMU zur Verwendung von PTB0 in beiden Betriebsarten veranlassen kann.

Die EIA-Register dienen zwei Zwecken: Wenn der Versuch zur Adreßumsetzung fehlschlägt, weil die Seite sich beispielsweise nicht im RAM befindet, legt man die virtuelle Adresse, die den Fehler verursachte, im EIA ab. Das Betriebssy-

stem kann diese Adresse wiedergewinnen, indem ein SMR-Befehl ausgeführt wird, außerdem kann die Ursache des Fehlers durch Untersuchung von MSR bestimmt werden.

Die zweite Funktion des EIA ist es, dem Benutzer das Löschen von Eintragungen im Cache zu ermöglichen. Der Cache enthält die 32 am häufigsten benutzten virtuellen Adressen mit den entsprechenden physikalischen Adressen. Dieser Speicher wird automatisch aktualisiert, ohne daß der Benutzer eingreifen muß. Allerdings wird ein Cache-Eintrag ungültig, wenn der dazugehörende Eintrag in der Seitentabelle durch Auslagern der Seite oder Änderung der Schutzebene verändert wurde. In solchen Fällen muß der Cache-Eintrag entfernt werden. Dies erreicht man ganz einfach durch Schreiben der virtuellen Adresse zum EIA-Register mit Hilfe des LMR-Befehls.

Wie Bild 7 zeigt, ist die MMU direkt mit dem Multiplex-Adreß-/Daten-Bus des Systems verbunden. Es gibt keine Bus-Konflikt-Probleme, weil die MMU ein aktives Low-Signal ausgibt (FLT), das die CPU vom Bus abschaltet, wenn die MMU auf das RAM zugreifen muß, um einen Seitentabelleneintrag zu holen.

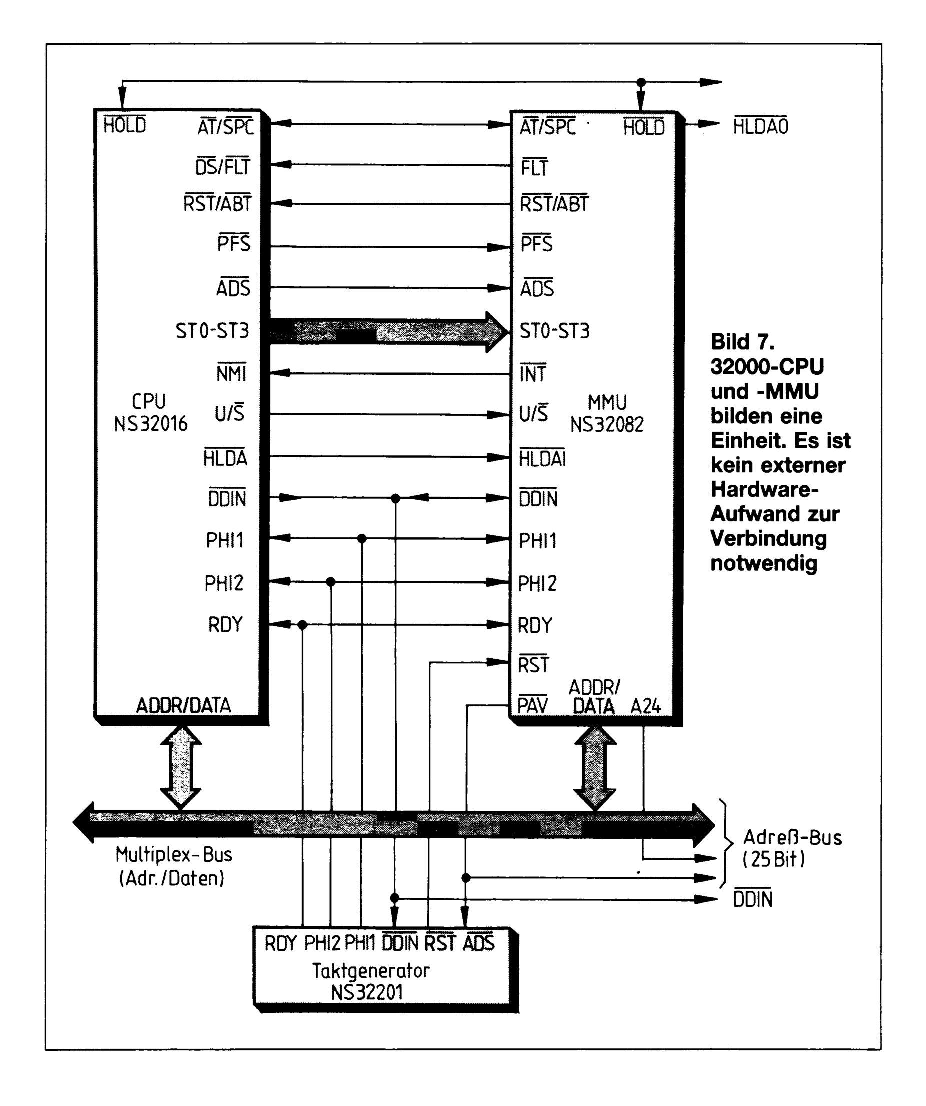


Bild 7. 32000-CPU und -MMU bilden eine Einheit. Es ist kein externer Hardware-Aufwand zur Verbindung notwendig

Alle CPUs der 32000-Familie können ohne Adreß-Umsetzung betrieben werden. Es ist daher notwendig, das Vorliegen einer Adreß-Umsetzung beim Einschalten des Systems zu melden. Dies ist ein zweistufiger Prozeß: Zunächst wird der AT/SPC-Anschluß während des Reset-Impulses auf Low gehalten. Dies hat zwei Effekte:

- Ein zusätzlicher Taktzyklus (TMMU) wird in alle Buszyklen eingefügt, außer bei Slave-Prozessor-Transfers. Während dieses Zyklus gibt die CPU ihre eigenen Adreßbus-Treiber frei, so daß die MMU die physikalische Adresse auf den Bus geben kann und das PAV-Signal (Physical Address Strobe) ausgibt, das die physikalische Adresse in die RAM-Adreßpuffer ablegt.
- Der DF/FLT-Anschluß schaltet von einem Data-Strobe-Ausgang zu einem Float-Kommando-Eingang um.

Das Anlegen des AT/SPC-Signals während des Reset-Vorganges sagt der CPU, daß die Adreßumsetzung erforderlich wird, außerdem, welche Hardware betroffen ist. Der zweite Schritt besteht deshalb darin, den CPU-Befehl „SETCFG" (Set Configuration) auszuführen, um festzustellen, daß die MMU-Befehle gültig sind.

Bild 8 zeigt die normalen zeitlichen Verhältnisse auf dem Bus für Lesezyklen, wenn sich die erforderlichen Adressen im MMU-Cache befinden. Während T1 gibt die CPU virtuelle

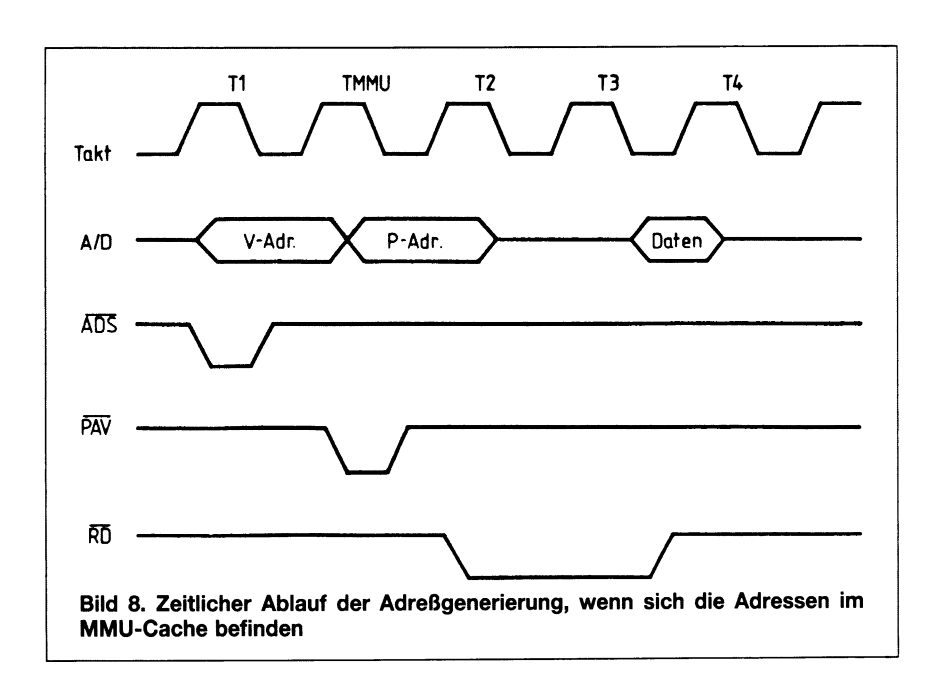


Bild 8. Zeitlicher Ablauf der Adreßgenerierung, wenn sich die Adressen im MMU-Cache befinden

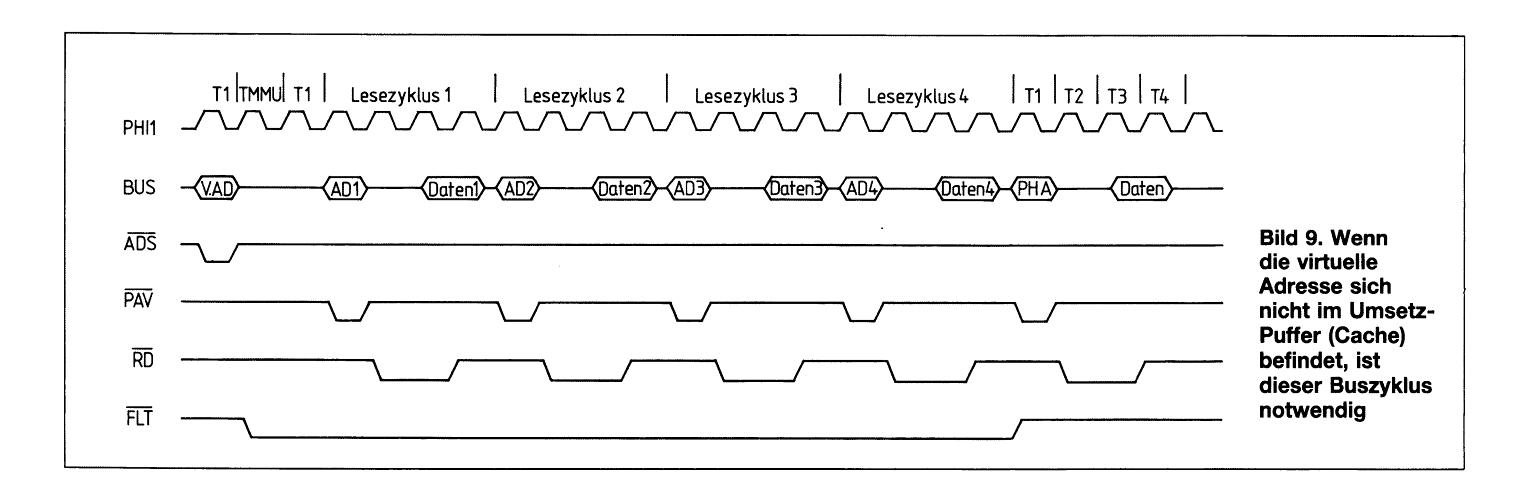


Bild 9. Wenn die virtuelle Adresse sich nicht im Umsetz-Puffer (Cache) befindet, ist dieser Buszyklus notwendig

Adressen aus, die in den MMU-Eingangspuffer gelangen, wenn der Adreßstrobe-Impuls ADS aktiv wird. Die CPU schaltet dann ihre Bustreiber ab, so daß die MMU die entsprechenden physikalischen Adressen während TMMU ausgeben kann. Der PAV-Impuls sagt dem Speicher, daß der Bus eine gültige physikalische Adresse enthält. Die MMU gibt anschließend den Bus wieder frei, so daß die CPU die Speicherdaten lesen kann. Jeder Speicherzugriff dieser Art, der etwa 98 % aller Speicherzugriffe darstellt, erfordert fünf Taktzyklen (bei nicht umgesetzten Systemen vier Taktzyklen).

Bild 9 zeigt die nächste wichtige Art von Buszyklen, die auftritt, wenn die virtuelle Adresse nicht mit einem Cache-Eintrag übereinstimmt, aber die Seite im RAM enthalten ist. In diesem Fall muß die MMU die physikalische Adresse von der Page-Tabelle im RAM holen, so daß an den FLT-Anschluß 18 Taktzyklen angelegt werden, während dieser die Doppelworte aufnimmt, die zur Ausführung der Umsetzung notwendig sind. Die MMU kann auch zwei Speicher-Schreibzyklen ausführen, wenn die Aktualisierung der M- und R-Statusbits in der Seitentabelle notwendig werden. Wenn die physikalische Adresse erst einmal vorliegt, gibt die MMU den Bus frei und die CPU arbeitet an der Stelle weiter, wo das Programm unterbrochen wurde.

Das wichtigste Merkmal dieses Mechanismus ist, daß die CPU nicht unterbrochen werden muß, wenn der Cache nicht die erforderliche Adresse enthält. Die MMU 68451 hat z. B. auch einen 32-Worte-Cache. Wenn allerdings die Adresse nicht im Cache ist, unterbricht die MMU die CPU, wobei der Programmzähler und verschiedene Datenregister im Stack zu speichern sind, bevor eine spezielle Trap-Routine ausgeführt werden kann. Diese Prozedur kann 50...100mal mehr Zeit als die 2 µs dauern, die für den Typ NS32082 notwendig sind.

Bis jetzt wurde davon ausgegangen, daß die MMU in der Lage ist, eine physikalische Adresse zu erhalten, allerdings gibt es auch eine Möglichkeit, bei der dies nicht der Fall ist. Entweder ist die Seite nicht im RAM oder die Pointer-Tabelle ist nicht im RAM oder die CPU hat versucht, auf eine geschützte Seite zuzugreifen. Jede dieser Bedingungen erzeugt einen sogenannten „Page-Fault". Ein Multi-Byte-Befehl kann z. B. eine Page-Grenze überschreiten, und die Seite, die die zweite Hälfte des Codes enthält, kann sich nicht im RAM befinden. Eine weitere Möglichkeit ist, daß die Instruktion eine Operandenadresse spezifiziert, die nicht im RAM enthalten ist. Der Abort-Interrupt-Mechanismus, den es nur bei der 32000-Familie gibt, erlaubt, daß die Befehle abgebrochen und neu gestartet werden können. Außerdem wird der Original-Programm-Zählerinhalt in ein Stack geladen, bevor die Steuerung dem Betriebssystem oder einer vom Benutzer definierten Service-Routine transferiert wird. Die Aufgabe der Korrektur der Seitenfehler wird vom Betriebssystem übernommen; die Hardware kann lediglich die Art des Fehlers spezifizieren. Das Betriebssystem bestimmt zunächst die genaue Natur des Problems, indem MSR untersucht wird. Wenn der Abbruch durch einen Übersetzungsfehler verursacht würde, liegen Bit 0 des MSR auf High-Potential und Bit 3...5 zeigen an, ob Befehle von einer Schutzebenen-Auslösung oder von einem ungültigen Tabelleneintrag in einer Ebenen-1-Seite herrührt, oder ob eine ungültige Eintragung in die Seitentabelle 2 vorgenommen werden sollte. Je nach Ursache des Fehlers will das Betriebssystem die Seite auswählen, die ausgelagert werden muß. Die Ausnahme dieser Regel ist, wenn die ursprüngliche Instruktion eine String-Instruktion war, weil dies einen unerwünschten Restart verursacht. String-Befehle sind daher Restart-Operationen.

Als Produkt Marketing Manager für Series 32000 ist **Uwe Krause (33)** verantwortlich für National's Produktstrategie bei der Serie 32000 Mikroprozessor Familie sowie der Erstellung einer entsprechenden Verkaufsstrategie.
Uwe Krause ist seit 1976 in der Semiconductor Branche beschäftigt und war dort zuständig für den Verkauf und Design-in Aktivitäten. Seit 1983 ist er verantwortlich für Series 32000 Marketing mit Standort Fürstenfeldbruck.

Uwe Krause

Geringe Stromaufnahme – hohe Verarbeitungsleistung

32-Bit-Mikroprozessor in CMOS-Ausführung

Unter der Bezeichnung NS32C016 bietet National Semiconductor die CMOS-Version der CPU 32016 an. Die ersten Bausteine der 32000-Familie wurden in XMOS-Technik, einem Hochleistungs-NMOS-Prozeß für hohe Geschwindigkeiten bei geringen Herstellungskosten, gefertigt. Die langfristige Ausrichtung, die ein wesentlicher Bestandteil der Strategie für die 32000-Familie darstellt, verlangt jedoch zukünftig die Verwendung der CMOS-VLSI-Technologie für eine Reihe von verbesserten Prozessoren, die derzeit in der Entwicklung sind. Der Typ 32C016 ist der erste in dieser Serie neuer Bausteine, die den Leistungsvorsprung der 32000-Familie auch in den nächsten 10 Jahren sicherstellen.

Überlegene CMOS-Technik

Weil die CPU 32C016 von der Funktion her identisch mit der „Original"-CPU 32016 ist – sie hat dieselbe Architektur und einen identischen Befehlssatz – ist in diesem Falle ein aussagefähiger Vergleich der Leistungsfähigkeit von fortgeschrittenen NMOS- gegenüber CMOS-Prozessoren möglich. Wie *Tabelle 1* zeigt, hat in bezug auf die Hauptleistungsmerkmale die CMOS-Technik Vorteile.

Als wesentlicher Faktor ist hervorzuheben, daß beide CPU-Bausteine mit derselben Taktfrequenz von 10 MHz arbeiten, obwohl Unterschiede in der Architektur in ihrer Wirkung unterschiedliche Taktfrequenzen bei weitem überwiegen. Wenn Mikroprozessoren mit verschiedenen Techniken, aber gleicher Architektur miteinander verglichen werden, so ist deren Leistungsfähigkeit direkt abhängig von der Taktfrequenz. Trotz dieser Gründe ist die CPU 32C016 dem Typ 32016 in der Gesamtleistung überlegen. Dieses Ergebnis wird manchen Entwickler überraschen, der üblicherweise die CMOS-Technik eher als besonders verlustleistungsarm einschätzt, und nicht als besonders schnell.

Die relativ geringe Verarbeitungsgeschwindigkeit der früheren CMOS-Prozessoren beschränkte deren Einsatz auf Gebiete, wo extrem geringe Verlustleistung und hohe Störfestigkeit die dominierenden Erfordernisse waren. Die gravierenden Verbesserungen, die besonders im Bereich der Zuverlässigkeit, Packungsdichte und auch in puncto Verarbeitungsgeschwindigkeit erzielt wurden, haben CMOS zur bevorzugten VLSI-Technologie gemacht, speziell auch deswegen, weil die ursprünglichen CMOS-Vorteile, geringe Verlustleistung und hohe Störfestigkeit, durchaus erhalten blieben.

Die Fortschritte bei den CMOS-Leistungsparametern sind ein unmittelbares Ergebnis der ständigen Geometrie-Verkleinerungen. Alle MOS-Prozesse profitieren von dieser Geometrie-Verkleinerung in einem größeren Maße als die Bipolarprozesse, aber bei CMOS ist der Gewinn noch größer als bei jedem anderen MOS-Prozeß. Motivation für die ständige Verkleinerung der Schaltungsgeometrie ist in erster Linie die Steigerung der Packungsdichte als Ergebnis daraus. Aber die Geometrie-Verkleinerung bewirkt darüber hinaus auch eine höhere Arbeitsgeschwindigkeit, geringe Verlustleistung und höhere Zuverlässigkeit.

Ein Beispiel: Wenn die Strukturlänge eines MOS-Transistors um den Faktor x verkleinert wird, so vermindert sich die auf dem Silizium belegte Fläche einer gegebenen Schaltung um den Faktor x^2, und die Zahl der Schaltungselemente, die auf einer vorgegebenen Fläche untergebracht werden können, steigt um den Faktor $1/x^2$. Zusätzlich zeigt sich, daß die Verlustleistung jedes

Tabelle 1. Vergleich der XMOS- und CMOS-Leistungsdaten

Parameter	XMOS (32016)	CMOS (32C016)
Temperaturbereich	0 bis 70 °C	−55 bis 125 °C
Versorgungsspannung	5 V, ±5 %	5 V, ±10 %
Stromaufnahme	300 mA max.	70 mA max.
Maximale Verlustleistung	1,5 W	0,35 W
Taktfrequenz	10 MHz	10 MHz

einzelnen Transistors um den Faktor 1/x wächst. Die Verzögerungszeit jedes Gatters reduziert sich um den Faktor x^2. Das Geschwindigkeits-/Leistungs-Produkt verbessert sich demgemäß ebenfalls um einen Faktor x.

Die Wirkung einer solchen Veränderung läßt sich gut darstellen, wenn man sie auf die Verkleinerung von 3 auf 2 µm bezieht. Bei x = 2/3 wird die Packungsdichte mehr als verdoppelt ($1/x^2 = 2{,}25$), während die Gatter-Verzögerungszeiten mehr als halbiert werden ($x^2 = 0{,}44$). Es entsteht jedoch ein 50%iger Zuwachs der Verlustleistung pro Schaltstruktur. Wenn alle Vorteile der erhöhten Packungsdichte ausgenutzt werden, umfaßt der Chip mehr als doppelt so viele Schaltstrukturen, so daß die Chip-Verlustleistung um nicht weniger als den Faktor 3,3 ansteigt. Der Anstieg der Gesamtverlustleistung pro Chip ist Hauptgrund dafür, warum eine Technik mit besonders geringer spezifischer Verlustleistung pro Gatter essentiell wichtig für die VLSI-Technik ist.

Das sind natürlich theoretische Ergebnisse, die nicht vollständig auf die Praxis übertragen werden können. Beispielsweise ist es nicht immer möglich, alle Vorteile der potentiellen Geschwindigkeitsverbesserung auszuschöpfen, weil RC-Verzögerungen auf den internen Bussen auftreten. Als das 32C016-Projekt gestartet wurde, war klar, daß die Fortschritte in der Fertigungstechnologie die funktionale Reproduktion eines XMOS-Prozessors in CMOS-Technik ohne Leistungseinbußen möglich machen würden.

Die Übereinstimmung von NMOS- und CMOS-Leistung während der vergangenen Jahre, verbunden mit höherer Komplexität des Entwicklungsprozesses bedeuten, daß die individuelle Kreativität des Entwicklers den Unterschied zwischen den beiden Technologien bestimmt. Zahlreiche Schaltungen innerhalb eines Mikroprozessors sind in Blöcken, wie Register, ROM, RAM, PLA und Bus-Strukturen organisiert, wo CMOS genauso schnell ist wie NMOS. In den Basis-/Logikfunktionen allerdings scheint NMOS doch noch schneller zu sein. Wie Tabelle 1 zeigt, ist jedoch das wesentliche Leistungsziel beim 32C016 realisiert, XMOS in bezug auf die Geschwindigkeit zu erreichen.

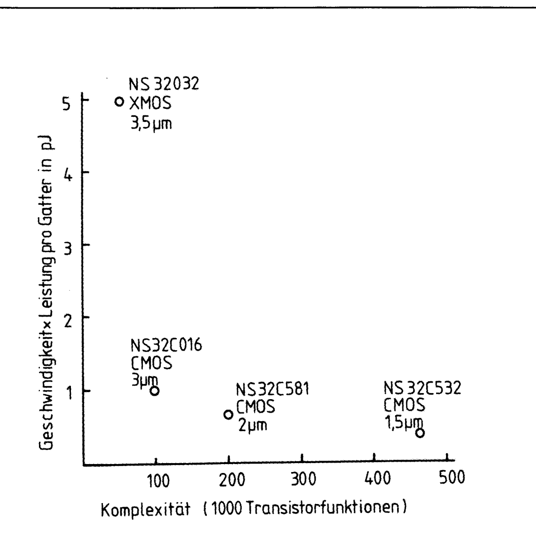


Bild 2. Die Entwicklung der Chip-Struktur führt von der 3,5-µ-XMOS-Technik bei der CPU NS32032 über die 3-µm- und 2-µm-Struktur bei den Bausteinen NS32C016 und NS32C581 zur 1,5-µm-CMOS-Struktur des NS32C532. Neben einer erheblichen Reduzierung des Produktes aus Geschwindigkeit und Verlustleistung pro Gatter um mehr als den Faktor 5 geht damit eine Steigerung der Komplexität von 60 K auf nahezu 500 K Funktionen pro Chip einher

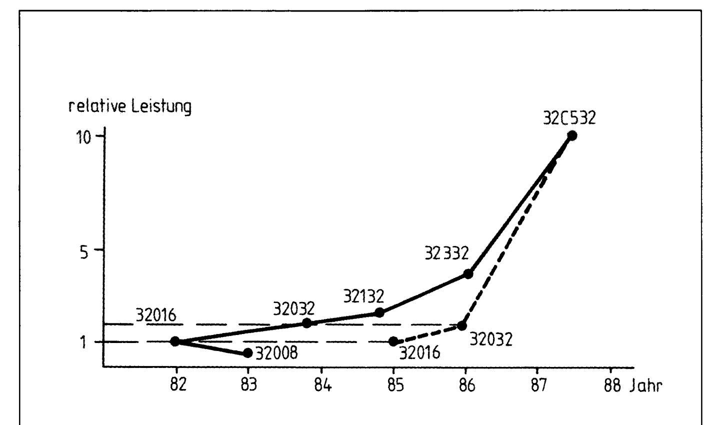


Bild 1. Die CPU 32C016 ist Teil einer langfristigen Strategie von National Semiconductor, die auf die Entwicklung von Hochleistungs-CMOS-Prozessoren mit einer geringen Verlustleistung und einer zu den bekannten XMOS- (NMOS-) Prozessoren kompatiblen Architektur zielt. Grundlage dieser Entwicklung ist die 1982 vorgestellte CPU 32016

Entwicklungsstrategie

Die CPU 32016 war ein geeignetes Objekt für das Entwicklungsprojekt. Deren moderne Architektur war bereits voll in XMOS implementiert, so daß es keine Logik- oder Systemprobleme gab. Die Entwicklungsanstrengungen konnten deshalb auf die Leistungs- und Zuverlässigkeitsziele konzentriert werden. Es war darüber hinaus gewährleistet, daß der neue Prozessor voll von dem Entwicklungssystem SYS32, dem Emulator ISE-16, den Betriebssystemen Genix und System-V sowie den C- und Pascal-Compilern unterstützt wird.

Der nächste Schritt war die Wahl eines geeigneten CMOS-Prozesses. Die Entscheidung fiel auf den 3-µm-Doppel-Poly-Prozeß von National Semiconductor, einer Weiterentwicklung des bewährten 5-µm-Prozesses. Entgegen früheren Befürchtungen, daß die Polysilizium-Verbindungen zu langsam wären, stellte sich heraus, daß die elektrischen Parameter den Anforderungen genügten. Untersuchungen zeigten, daß kurze Poly-Verbindungen (1000 µm oder kleiner) die Gatter-Verzögerungszeiten im allgemeinen nicht signifikant beeinflussen. Bei den internen Bussen und anderen kritischen

Signalwegen ließen sich potentielle Geschwindigkeitsprobleme durch den gezielten Einsatz von Metallverbindungen beseitigen. Darüber hinaus zeigte sich, daß bei sehr langen Verbindungswegen die kapazitive Last des Treibers eine wesentlich größere Geschwindigkeitsbegrenzung bedeutete, als die RC-Verzögerung durch das Poly-Silizium. Aus diesem Grunde wurde den Verbindungen zwischen den einzelnen Chip-Sektionen und dem Gesamt-Chip-Layout große Aufmerksamkeit geschenkt, bevor mit dem detaillierten Logik-Design begonnen wurde.

Die bei der Chip-Planung angesteuerten Ziele waren, in Rangfolge ihrer Priorität:
- Optimierung der Leistung,
- Schaffung einer optimalen Chip-Struktur,
- Minimierung der Chip-Fläche.

Hauptstrategie für die Entwicklung war, der XMOS-Struktur soweit wie möglich zu folgen, nicht nur, um zu verhindern, daß das Rad zum zweiten Mal erfunden wird, sondern auch, um einen möglichst großen Nutzen aus der Arbeit zu ziehen, die während der Entwicklung einer effizienten und fehlerfreien Architektur gemacht wurde.

Bei der Entwicklung der Chip-Struktur galt das wesentliche Augenmerk der Aufteilung in funktionale Blöcke und den Verbindungen zwischen den Sektionen. Die XMOS-Struktur diente als Vorbild, obwohl es in Details des Logik-Design erhebliche Unterschiede zwischen den beiden Bausteinen gibt. Es wurde nicht der Versuch unternommen, ein exaktes Spiegelbild der XMOS-Struktur zu schaffen. Das geschah teilweise deshalb, weil einige Funktionen, wie z. B. NOR-Gatter, die häufig in XMOS verwendet werden, nicht so einfach in CMOS implementierbar sind. Noch wesentlicher war jedoch, daß eine direkte Übertragung der Schaltungslogik den effizienten Einsatz der CMOS-Strukturen nicht zuläßt. Konsequenterweise wurde die XMOS-Logik in großem Umfange umstrukturiert, um die Vorzüge des CMOS-Prozessors maximal nutzen zu können.

Obwohl die Taktgeschwindigkeit ein Haupt-Leistungsparameter darstellt, ist sie nicht der einzig wichtige Parameter, denn die CPU 32C016 bietet alle traditionellen CMOS-Vorzüge. Mit 10 MHz Taktfrequenz ist CMOS nicht länger *nur* eine „Mikrowatt-Technologie", jedoch bietet sie auch weiterhin eine signifikant geringere Verlustleistung als Bipolar- oder NMOS-Prozesse. Ein unmittelbarer Nutzen ist, daß kleinere und billigere Stromversorgungen eingesetzt werden können. CMOS erlaubt auch die Entwicklung von leistungsfähigen portablen Computern und ähnlichen Produkten, die mit Batterien auskommen.

Die geringe Verlustleistung hat zum Ergebnis, daß CMOS-Mikroprozessoren in Geräten und Systemen einsetzbar sind, wo NMOS-Prozessoren nicht arbeiten können. In Automobilanwendungen z. B. ist der Platz innerhalb des Armaturenbrettes begrenzt, so daß elektronische Bauelemente im Motorraum untergebracht werden müssen. Mit Temperaturunterschieden von Frost bis zu kochender Hitze und einem hohen Grad an elektrischen Störungen ist das Auto eine der rauhesten Umgebungen, die man sich vorstellen kann. CMOS ist die einzige Technologie, die hierfür effektiv eingesetzt werden kann.

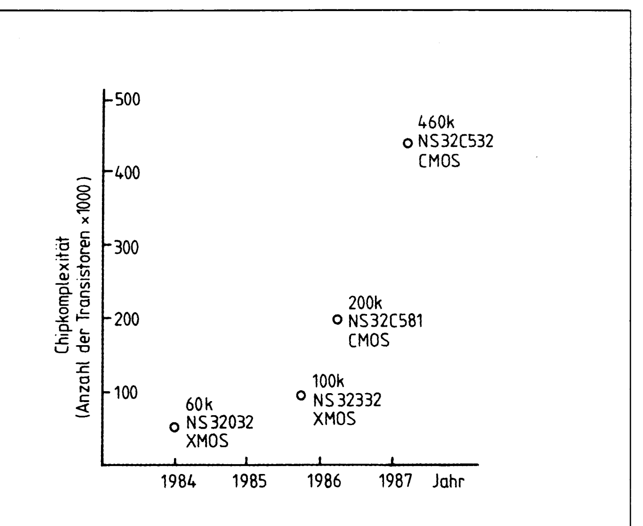


Bild 3. In nur drei Jahren, von 1984 bis 1987, gibt es, dank der fortschrittlichen CMOS-Technik von National Semiconductor, eine rasante Entwicklung bei der Anzahl der Transistorfunktionen pro Chip. Meilensteine auf diesem Weg sind die Bausteine NS32032, NS32332, NS32C581 und NS32C532.

Geringe Verlustleistung führt auch direkt zu höherer Zuverlässigkeit. Die Tatsache, daß ein Baustein für einen bestimmten Arbeitstemperaturbereich spezifiziert ist, bedeutet nicht, daß seine Arbeits-/Lebenszeit über diesen Bereich konstant ist. Es ist weitgehend bekannt, daß die Arbeits-/Lebenszeit eine Funktion der Chip-Temperatur ist. Weil die CPU 32C016 „kälter" als die XMOS-Version arbeitet, kann sie bei höheren Umgebungstemperaturen eingesetzt oder mit längerer Lebensdauer bei derselben Temperatur betrieben werden.

Die maximale Verlustleistung von 350 mW des Bausteins 32C016 ist sehr niedrig, auch für CMOS-VLSI. Ein kürzlich vorgestellter CMOS-Mikroprozessor hat z. B. eine maximale Verlustleistung von 2 W. Der Hauptgrund für den geringen Verlustleistungswert beim 32C016 ist die „saubere" Architektur, die mit rund 60 000 Transistoren auskommt. Bringt man diesen Wert in einen Bezug, so geht man davon aus, daß das Gehäuse einen thermischen Widerstand von 30 K/W hat. Bei 2 W Verlustleistung wird die Chip-Temperatur etwa 60 K über der Umgebungstemperatur liegen. Bei einer Verlustleistung von lediglich 350 mW reduziert sich diese Temperaturdifferenz auf nur 10,5 K.

Störfestigkeit

Auch wenn der Mikroprozessor bei Raumtemperatur betrieben und von einer geregelten Stromversorgung

versorgt wird, haben CMOS-Mikroprozessoren aufgrund ihrer größeren Störfestigkeit einen bedeutenden Vorteil. Störungen haben im Prinzip zwei Ursachen: Einerseits entstehen sie extern durch elektromagnetische Impulse oder als Transienten auf der Versorgungsleitung. Diese Art von Störungen treten nach dem Aufbau des Systems auf. Es gibt jedoch zum Zweiten auch intern erzeugte Störungen durch unbeabsichtigte Koppeleffekte von einem Teil des Systems auf ein anderes.

Eines der Probleme, die Entwickler heute zu lösen haben, ist die Notwendigkeit, immer höhere Computerleistung auf einer Leiterplatte unterzubringen, deren Größe konstant bleibt. Die Schaltungsdichte auf den Leiterplatten nimmt deshalb ständig zu und hat heute einen sehr hohen Grad erreicht. Deswegen müssen Bauelemente, die elektrische Störungen erzeugen, zwangsläufig nahe empfindlichen Bausteinen plaziert, die Breite der Masseleitungen muß verkleinert werden und es ergeben sich oft lange parallele Busleitungen.

Die heute zur Verfügung stehenden CAD-Werkzeuge und Entwicklungstechniken sind nicht geeignet, das Problem der induktiven und kapazitiven Störspannungen beim Leiterplatten-Layout zu berücksichtigen, so daß Störungen zu einem erheblichen Problem bei der Systementwicklung werden können. Die größere Störfestigkeit von CMOS-Bausteinen minimiert dieses Problem und erlaubt es, Leiterplatten mit höherer Pakkungsdichte aufzubauen oder mit höherer Geschwindigkeit zu betreiben. Der Aufbau von Prototypen ist ebenfalls einfacher, wenn CMOS-Bauelemente eingesetzt werden. Wirewrap-Prototyp-Leiterplatten z. B. haben eine große Zahl von Streuinduktivitäten und -kapazitäten, die in der Serienversion dann nicht mehr vorhanden sind. Die größere Störfestigkeit von CMOS-Schaltungen bedeutet in diesem Falle, daß die Leistungsdaten des Prototyps denen des Seriensystems sehr viel näher kommen als bei anderen Techniken.

Der Typ NS32C016 hat TTL-kompatible Eingangsspannungspegel, so daß damit eine direkte Austauschbarkeit in vorhandenen Applikationen besteht, wobei nur Anschluß 29 mit +5 V verbunden werden muß.

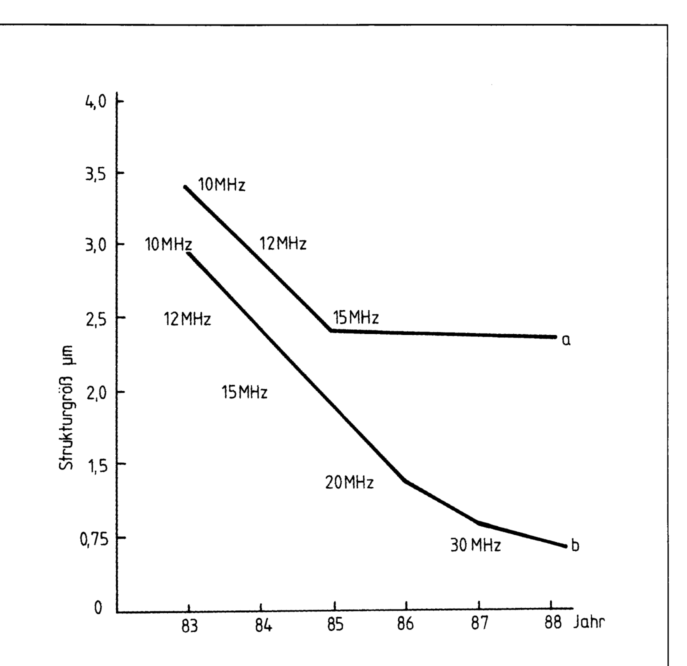


Bild 4. Mit der Verkleinerung der Halbleiterstrukturen ist eine rapide Steigerung der Taktfrequenz möglich. Bei den CMOS-CPUs werden dreifach höhere Taktfrequenzen angestrebt (von 10 MHz auf 30 MHz).

Speicherverwaltungs-Einheit NS32382

Die Weiterentwicklung der MMU für die Mikroprozssor-Familie 32000 diente in erster Linie dazu, auch den Bereich des externen 32-Bit-Datenbusses und der Adressierung von 4 GByte abdecken zu können. Der Trend zu einem 32 Bit breiten externen Daatenbus beruht auf der Tatsache, daß der Datenaustausch mit dem Hauptspeicher beim Übergang von 16 Bit auf 32 Bit verdoppelt werden kann, daß die Taktfrequenz der CPU erhöht oder die Zugriffszeit der Speicherelemente verkürzt werden muß.

Die Leistungssteigerung wird hier durch eine Erhöhung der Kosten flür den externen Datenbus – Verdopplung der Datenbustreiber – erkauft. Die Verdopplung der Speicherbussteine ist in den meisten Applikationen ein kleineres Problem, da die zum Einsatz kommenden Speicherkapazitäten ein Vielfaches der verwendeten RAM-Bausteine ist.

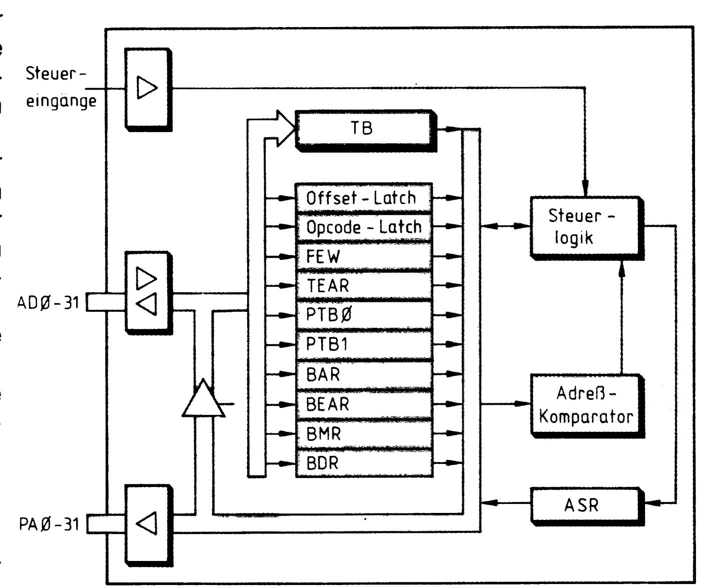
Obwohl die MMU NS32382 nach den gleichen Prinzip wie der Typ NS32082 funktioniert, gibt es kleine Unterschiede in der Implementierung. Der Typ NS 32382 wird in einem 122 poligen Pin-Grid-Array geliefert (NS32082 ist in einem 48 poligen DIP-Gehäuse untergebracht).

Die Erhöhung der Auschlußzahl beruht im Wesentlichen auf der geänderten Implemementierung des physikalischen Adreßbusses, der jetzt nicht mehr mit dem virtuellen Adreß- und dem Datenbus der CPU im Multiplex umgeschaltet wird. Diese Änderung ermöglichte eine Vereinfachung des Aufbaus für die Adreßtreiber und deren Steuerlogik auf der MMU. Außerdem verbesserte sich ebenfalls das Zeitverhalten der Adreßumsetzlogik und ermöglichte so die Umsetzung der virtuellen Adressen in physikaische Adressen in nur einer Taktperiode bei 15 MHz.

Eine weitere Änderung betrifft die Seitengröße, die von 512 Bytes auf 4096 Bytes erhöht wurde. Diese Anpassung erfolgte, um die Verwaltung der Seiten weiterhin in einer zweistufigen Seitentabelle durchzuführen zu können. Die Vergrößerung der Seiten um den Faktor 8 trägt auch dem gestiegenen Datendurchsatz bei den neuen Massenspeichern Rechnung. Hier sind die Bit-Übertragungsfrequenzen von ehemals 2...3 MBit auf beachtliche 20...24 MBit angestiegen.

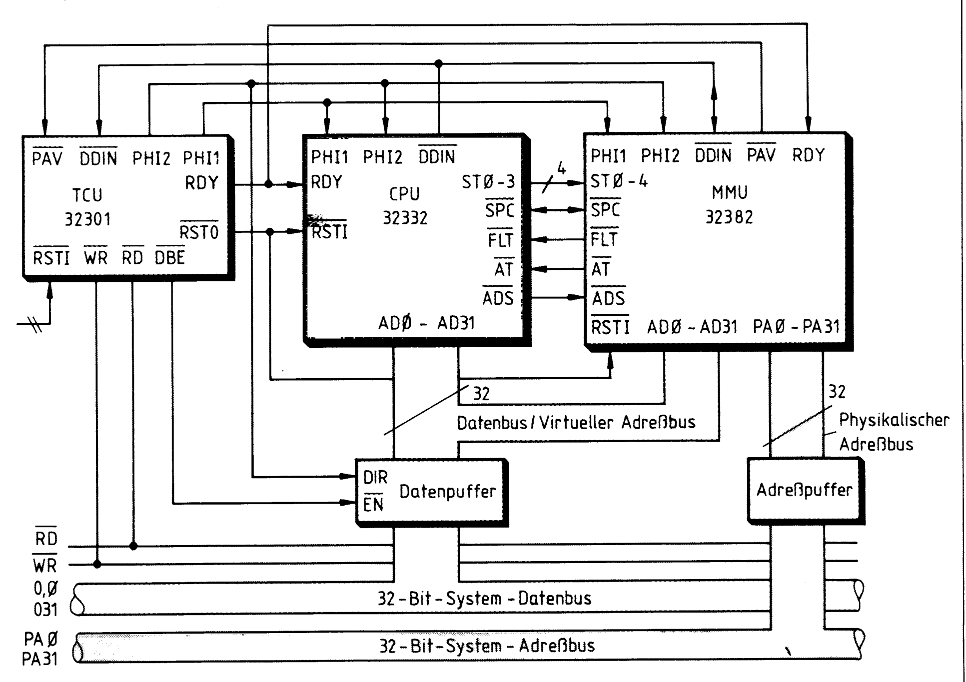
Der Adressen-Umsatzspeicher „Translation Buffer" (TB) speichert bis zu 32 Umsetzvektoren, die eine direkte Umsetzung von virtuellen Adressen in physikalische Adressen erlaubt.

Gleichzeitig mit der Umsetzung erfolgt eine Überprüfung der zu der Seite gehörenden Statusinformationen. Wird keine Schutzfunktion verletzt, so erfolgt die Ausgabe der physikalischen Adresse nach ca. 60 ns über den physikalischen Adreßbus.
Das „Offset-Latch" übernimmt die niederwertigen 12 Bit der virtuellen Adresse und gibt sie über den physikalischen Adreßbus aus. Diese 12 Bit definieren den Offset von der Basis-Adresse einer Seite im virtuellen und physikalischen Adressraum und wird daher durch die Umsetzung nicht beeinflußt.
Im „Opcode-Latch" hält die MMU den Befehlcode, den die CPU am Anfang des Slave-Protokolls an die MMU übergibt. Die nachfolgenden Daten werden von der MMU dann in die entsprechenden Register abgelegt, oder aus diesen Registern gelesen und an die CPU übergeben.
Die Funktionen der MMU wird durch den Eintrag in dem Register FEW (Feature Enable Word) bestimmt. Nach einem Reset sind alle 16 Bit in diesem Register auf Null gesetzt und die MMU erscheint als nicht vorhanden im System. In der Regel beginnt man erst nach Abschluß der System-Initalisierung oder nach erfolgreichem Start des Betriebssystems mit der Programmierung der MMU. Der Programmierer bestimmt somit, wann und wie Seitenumsetzungen sowie Breakpoints in der MMU zur Wirkung kommen.
Ist die Umsetzung einer virtuellen Adresse in eine physikalische für die MMU nicht erfogreich durchzuführen, so sendet die MMU ein Abbruchsignal „Abort Translation" (AT) an die CPU und sichert die virtuellen Adresse in dem „Translation Error Address Register" (TEAR). Eine Unterbrechungs-Service Software kann nun diesen Eintrag auslesen und die notwendigen korrektiven Maßnahmen ergreifen.
Die MMU besitzt zwei Seitentabellen-Basisadressregister „Page Table Base Register" (PTBO) und (PTB1). Diese Register halten die Anfangsadressen der Seitentabellen der ersten Stufe eines Programmes. Die Seitentabelle der ersten Stufe muß für jedes Programm komplett im Speicher vorhanden sein, bevor dieses gestartet wird. Die MMU ist damit in der Lage, für zwei unabhängige Programme die notwendige Adreßumsetzung durchzuführen. In der Regel befinden sich PTBO die Seitentabelle des zur Zeit ablaufenden Anwenderprogrammes.
DAS „Breakpoint Address Register" (BAR) erlaubt das Einkreisen von Programmfehlern durch Setzen von Breakpoints. In das Register wird die gewünschte 32-Bit-Adresse geschrieben und durch Setzen der Bits im FEW-Register die gewünschte Funktion aktiviert. Der Breakpoint kann auf das Programm in PTBO oder PTB1, Befehlszugriff und/oder Datenzugriff gesetzt werden. Beim Datenzugriff kann als zusätzliche Unterbrechungskondition das Lesen und/oder Schreiben spezifiert werden.
Mit dem „Breakpoint Mask Register" (BMR) besteht die Möglichkeit, die festgelegten Unterbrechungsbedingungen auf einen Block im virtuellen Adreßbereich anzuwenden. Mit dieser Option sind Datenmanipulatinen in Arrays und Tabellen leichter zu Überwachen als mit Unterbrechung auf einzelne Adressen.



Das „Breakpoint Data Register" (BDR) beinhaltet die virtuelle Adresse die die Unterbrechung auslöste. Diese Information ist dann besonders nützlich, wenn vom BMR ein Speicherblock selektiert wurde und nach erfolgter Unterbrechung die genaue Adresse für weitere Untersuchung benötigt wird.
Im „Abort Status Register" (ASR) sichert die MMU bei Auftreten eines Fehlers die für eine Auswertung wichtigen Statusinformationen. Die CPU wird bei der Behandlung des MMU Abbruches auf jeden Fall erst das ASR auslesen und daraufhin notwendigen Aktionen einleiten. Bei Bedarf werden auch die anderen Register zur Klärung der Abbruchursache ausgelesen.
Zusätzlich zu den bereits erwähnten Merkmalen ermöglicht die neue MMU den Aufbau eines Systems mit einem virtuellen Daten- und Befehls-Cache ohne einfügen von Wait-Zyklen. Auch bei der maximalen Taktfrequenz von 15 MHz bleibt dann für den Cache-Komperator noch eine Reaktionszeit von 115 ns.

Uwe Krause



Uwe Krause

32000-Familie unterstützt modulare Programmierung

In den vergangenen zehn Jahren führte die schnelle Weiterentwicklung der Mikroprozessor-Hardware zu einer zunehmenden Komplexität bei der Software. 32-Bit-Prozessoren der jüngsten Generation haben wenig Ähnlichkeit mit den ursprünglichen 4-Bit-CPUs; kleine monolithische Programme, die in Maschinencode geschrieben sind, wurden schon seit einiger Zeit von großen und komplexen Programmen ersetzt, die üblicherweise von einem umfangreichen Programmiererteam geschrieben werden und parallel in einer höheren Sprache arbeiten. Auf diese Weise erreicht man eine effiziente Software-Entwicklung.

Modulare Programmierung ist heute die bevorzugte Technik zum Abwickeln größerer Softwareprojekte. Das Aufteilen einer komplexen Anwendung in kleine Tasks, die wiederum weiter unterteilt sind, ermöglicht es, daß große Programme systematisch erstellt werden können. Programme, die auf diese Weise entwickelt wurden, arbeiten mit einer größeren Wahrscheinlichkeit korrekt. Sie sind darüber hinaus besser zu warten und zu dokumentieren. Normalerweise ist es auch möglich, die Haupt-Task in eine Anzahl von eigenständigen Modulen aufzuteilen, die sich parallel entwickeln lassen. Dies ist häufig der einzige Weg, mit dem sich große Programme in realistischen Zeiten schreiben lassen.

Ein anderer Vorteil, der sich aus der Aufteilung eines Anwendungsprogrammes in kleine, unabhängige Module ergibt, ist die Tatsache, daß es häufig möglich ist, diese Module für andere Anwendungen leicht umzuschreiben. Theoretisch kann ein Benutzer eine ganze Bibliothek von Modulen aufbauen, die sich als Bausteine für weitere Programme nutzen lassen. Wenn man einen Schritt weitergeht, ist es denkbar, diese Module in ROMs abzulegen, so daß sich große Programme auch mit Hilfe von hardwareähnlichen Komponenten zusammensetzen lassen.

Den meisten Mikroprozessorarchitekturen fehlen allerdings die wichtigsten Merkmale, die notwendig sind, um diese Idee in die Praxis umzusetzen. Egal ob sich diese Module in einem ROM oder auf einer Platte befinden: Softwaremodule, die kompiliert und assembliert sind, auch „Objekt-Module“ genannt, sind üblicherweise als relocatibler Objektcode gespeichert. Wenn eine Anzahl von Objektmodulen in den Speicher zur Ausführung geladen werden, ist ein Linking-Editor-Programm erforderlich, um alle unübersetzten Adressen in absolute Adressen umzuformen. Wenn diese unübersetzten Adressen innerhalb des derzeit bearbeiteten Moduls liegen, bezeichnet man den Prozeß als „Relocation“; wenn die Adressen sich auf Variable oder Prozeduren in anderen Modulen beziehen, heißt dieser Prozeß „Linking“. Bei großen Programmen, die sich auf Magnetplatten befinden, sind Linking und Relocation aufwendige und zeit-konsumierende Aktivitäten; bei Programmen, die sich in ROMs befinden, lassen sich die Adressen nicht verändern.

Architektur entsprechend ausgelegt

Beim Entwurf der Serie 32000 erkannte man, daß die effiziente Softwaregenerierung genauso wichtig ist wie die effiziente Softwareausführung. Daher enthält die Architektur der Serie 32000 Mechanismen, die Module und Prozeduren unterstützen, zwei Schlüsselelemente der strukturierten Softwareentwicklung. Prozeduren sind die elementaren Software-Bausteine. Bei der Programmausführung korrespondiert die Prozedur mit einem Unterprogrammaufruf. Die 32000-Architektur bietet allerdings auch Register und Befehle, die die Zuordnung von Speicher für Prozedur-Parameter und lokale Variable verwaltet.

Obwohl der Begriff Modul sehr häufig ziemlich ungenau definiert benutzt wird, hat er für 32000-Systeme eine spezielle Bedeutung. Ein Modul, das normalerweise eine Anzahl von Prozeduren enthält, besteht aus drei Komponenten: Programmcode, statische Daten und

Link-Tabelle. Der Programmcode enthält die konstanten Daten des Moduls (Literals) sowie den eigentlichen Programmcode. Unter statischen Daten des Moduls versteht man globale Variablen und Daten, das heißt Daten, auf die alle Prozeduren innerhalb eines Moduls zugreifen können. Die Linktabelle, die später genauer beschrieben wird, benutzt man, wenn Variablen und Prozeduren in anderen Modulen, auf die zugegriffen werden kann, untergebracht werden.

Idealerweise sollte es möglich sein, Module im ROM zu speichern, und die ROM-Module beliebig im Speicher zu positionieren. Bevor hier näher beschrieben wird, wie die traditionellen Relokations- und Linking-Programme minimiert werden, ist es notwendig, bestimmte Merkmale der 32000-Architektur zu betrachten.

Die Register

Bild 1 zeigt das 32000-Programmiermodell. Die 15 Register, die jeweils 32 Bit breit sind und auf die ein Benutzer zugreifen kann, sind in zwei Gruppen geteilt. Acht davon sind universelle Register, die sich als Index-Register, Daten-Register oder Adreß-Pointer ohne jede Beschränkung benutzen lassen. Von den übrigen sieben Registern haben der Programmzähler, die Interrupt-Basis sowie die Stack-Pointer Spezialfunktionen. Das statische Basis-Register wird als Pointer auf statische oder globale Variablen einer Prozedur oder eines Moduls benutzt, während der Frame-Pointer zum Zugriff auf die Parameter und lokalen Variablen einer Prozedur dient.

Das Netz-Register in zwei 16-Bit-Bereiche unterteilt. Die obere Hälfte ist das Prozessor-Status-Register, die untere Hälfte stellt das Modul-Register dar. Dieses enthält die Adressen eines 16-Byte-Blocks im Speicher, der

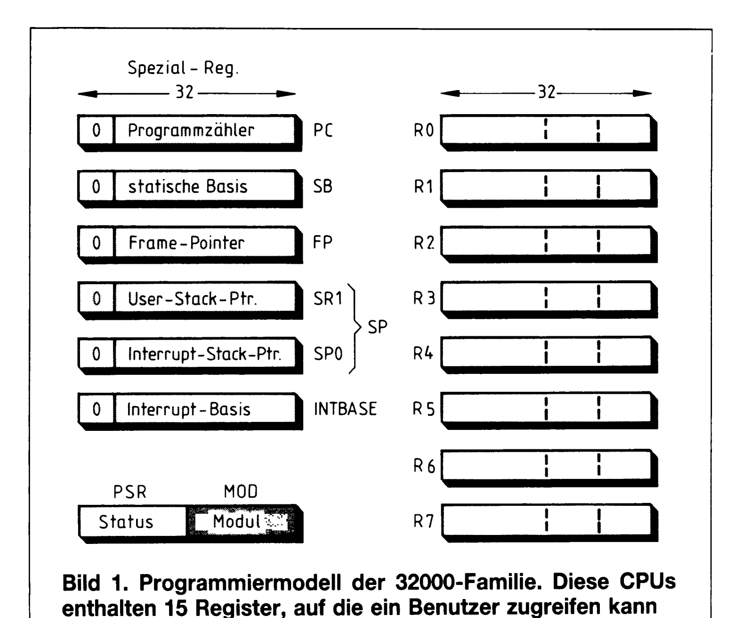


Bild 1. Programmiermodell der 32000-Familie. Diese CPUs enthalten 15 Register, auf die ein Benutzer zugreifen kann

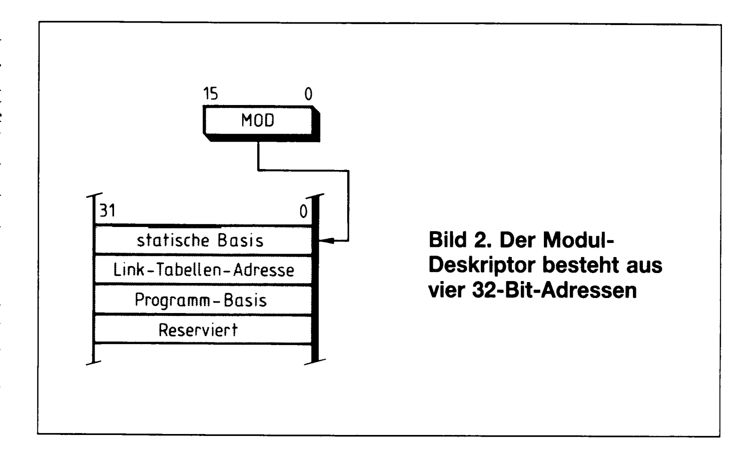


Bild 2. Der Modul-Deskriptor besteht aus vier 32-Bit-Adressen

das derzeit ausführende Modul definiert. Dieser Block, der auch „Modul-Deskriptor" genannt wird, wird als vier 32-Bit-Adressen interpretiert (*Bild 2*). Eine dieser Adressen ist für zukünftigen Gebrauch reserviert, während die anderen auf den Code des Moduls zeigen sowie die statischen Datenblöcke und dessen Linktabelle. Es ist zu beachten, daß alle Modul-Deskriptoren innerhalb der ersten 64 KByte des virtuellen Speichers liegen müssen, weil das MOD-Register eine Länge von 16 Bit hat.

Adressierarten

Zu den wichtigsten Aspekten der Mikroprozessorarchitektur zählen die unterstützten Adressierarten. Die Architektur der Serie 32000 bietet die Standard-Adressierarten, z. B. Register, Immediate, Absolut und Register-Relativ. Darüber hinaus gibt es noch fünf Adressierarten, die speziell zur Unterstützung von Hochsprachen-Compilern entwickelt wurden, und drei davon (Speicherbereich, Speicher-Relativ und extern) sind für modulare Programmierung von besonderer Bedeutung.

In der Speicherbereich-Betriebsart ist eines der Spezial-Register (PC, SP, SB oder FP) als Basis-Pointer definiert. Ein vorzeichenbehaftetes Displacement von bis zu 30 Bits wird dem Inhalt des gewählten Registers hinzugefügt, um die Operandenadresse zu erhalten. Die speicher-relative Betriebsart stellt eine zweite Ebene dar. Das erste Displacement wird dem Inhalt des bestimmten Basis-Register, in ähnlicher Weise wie bei der Speicher-Slice-Betriebsart hinzugefügt. Die sich daraus ergebende Adresse ist allerdings noch nicht die Operandenadresse, sondern die Adresse eines Pointers, zu dem ein zweites Displacement addiert wird, um die endgültige Operandenadresse zu erhalten.

Die externe Betriebsart ermöglicht es, daß Module ohne Linkage-Editierung relociert werden. Damit ist es möglich, daß Variable und Prozeduren, die außerhalb der derzeit ausgeführten Module auch zugreifbar sind, was mit Hilfe der Link-Tabelle des Moduls geschieht. Dies ist ähnlich wie bei der speicher-relativen Betriebsart, bei der zwei Displacement benutzt werden, unter-

scheidet sich aber darin, daß die erste Basisadresse die Link-Tabelle des derzeit in Arbeit befindlichen Moduls darstellt.

Diese leistungsfähigen relativen Adressierarten und die Symmetrie des Befehlssatzes verursachen keine Probleme bei der Relokation in 32000-Systemen. Innerhalb eines Moduls sind alle Datenzugriffe sowie die gesamten Unterprogrammaufrufe und Programm-Verzweigungen mit Hilfe von Displacements zu spezifizieren, die aus dem Programmzähler, statischen Basis-Register oder Frame-Pointer stammen. Der sich dabei ergebende Code ist positions-unabhängig; er kann sich irgendwo im Speicher befinden und läuft trotzdem korrekt ab.

Die derzeit verfügbaren 32000-CPUs arbeiten mit einem 24-Bit-Adreßbereich, während die Displacements, die in den relativen Adressierarten spezifiziert sind, einen vorzeichenbehafteten 30-Bit-Wert annehmen können. Konsequenterweise läßt sich positionsunabhängiger Code immer produzieren, unabhängig von der Größe der Module. Es ist zu beachten, daß von der Speicherverwaltungseinheit unterstützte virtuelle Speicher mit Demand-Paged-Zugriff die Relocation von Modulen ohne Editierung im virtuellen Adreßbereich sowie im physikalischen Adreßbereich möglich machen.

Linken vereinfacht

Wenn ein Programm mehr als ein Modul umfaßt, ist es wahrscheinlich, daß gewisse Prozeduren von Zeit zu Zeit Zugriff auf Variablen des anderen Moduls benötigen oder andere Prozeduren der anderen Module aufrufen. Um den Link-Vorgang von Modulen zu vereinfachen, sind in die Architektur der 32000-Familie Einrichtungen für externe Bezüge eingebaut. Unter der Voraussetzung, daß jeder externe Zugriff über die Link-Tabelle des Moduls erfolgt, ist der Programm-Code eines Moduls nicht abhängig von der Laufzeit-Position anderer Module, so daß sich dieser Code in einem ROM unterbringen läßt, falls dies erforderlich ist.

Die beiden wichtigsten Einrichtungen in dieser Beziehung sind die externe Adressierart für den Zugriff auf externe Variablen sowie der Befehl CXP (Call external Procedure) zusammen mit der entgegengesetzten RXP-Instruktion (Return vom external Procedure). In beiden Fällen wird das MOD-Register in Zusammenhang mit der Link-Tabelle benutzt, um in bezug auf die Laufzeit absolute Adressen zu liefern.

Anfang	31	0	
0	Absolute Adresse		(Variable)
1	Absolute Adresse		(Variable)
2	Offset	Modul	(Prozedur)

Bild 3. Aufbau der Link-Tabelle. Es handelt sich um eine Liste absoluter Adressen, die alle externe Variablen und Prozeduren widerspiegeln

Bild 3 zeigt, wie eine Link-Tabelle aufgebaut ist. Grundsätzlich handelt es sich um eine Liste von absoluten Adressen, die allen externen Variablen und erforderlichen Prozeduren entsprechen. Sie wird normalerweise vom Linking-Editor aufgebaut, wenn Module in den Speicher geladen werden. Sie kann aber auch genauso gut im ROM abgelegt sein, wenn das System nur auf der Basis von ROMs aufgebaut ist.

Es gibt zwei Arten von Eintragungen in eine Link-Tabelle. Für externe Variablen gibt es ganz einfach die absoluten 32-Bit-Adressen der Variablen. Für externe Prozeduren ist die Eintragung in zwei 16-Bit-Worte unterteilt. Das niederwertige Wort ist die Adresse des Modul-Deskriptors für das Modul, das die externe Prozedur enthält. Das höherwertige Wort ist ein Offset, der die Adresse für den Prozedur-Anfangspunkt in bezug

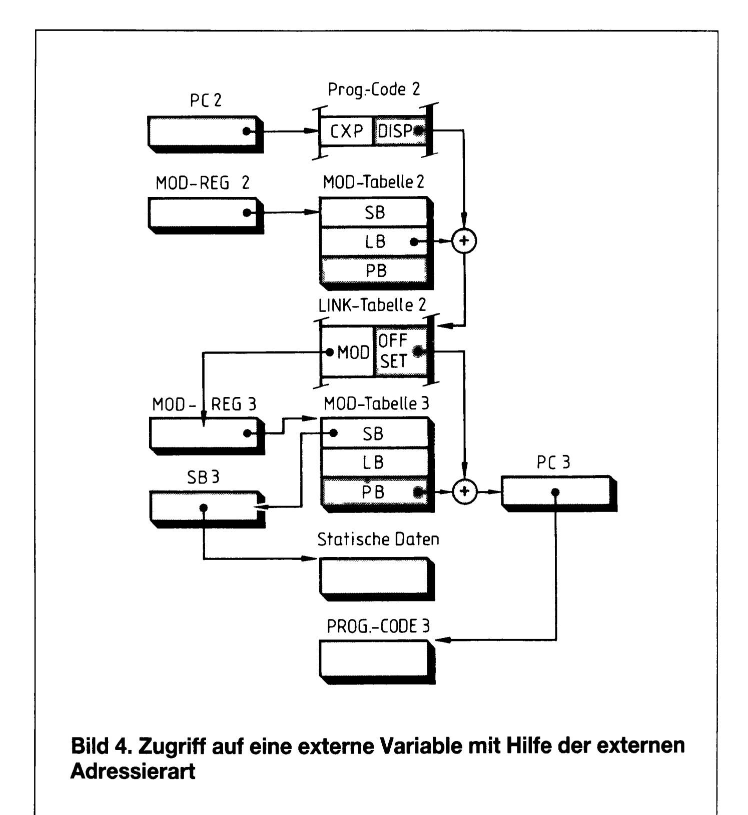


Bild 4. Zugriff auf eine externe Variable mit Hilfe der externen Adressierart

auf den Start des Programm-Code-Blocks des externen Moduls angibt. Obwohl ein 16-Bit-Offset bedeutet, daß der Anfangspunkt innerhalb der ersten 64 KByte des Modul-Codes liegt, ist dies keine wichtige Beschränkung, weil man am Beginn von sehr großen Modulen eine Tabelle von relativen 30-Bit-Verzweigungen plazieren kann.

Modulorientierte Mechanismen

Bild 4 zeigt einen Zugriff auf eine externe Variable mit Hilfe der externen Adressierart. Die Variable wird als

Code des Moduls definiert, indem dessen Position in der Link-Tabelle angegeben wird. Dies wird von einem Displacement im Befehlscode spezifiziert. Die absolute Adresse der Variable wird folgendermaßen berechnet:

1. Das MOD-Register zeigt auf den Modul-Deskriptor des derzeit abgearbeiteten Moduls. Von hier aus wird die Startadresse der Link-Tabelle geholt.
2. Das Displacement, das die Position der Variablen innerhalb der Link-Tabelle definiert, wird zur Startadresse addiert, um eine Basisadresse für die externe Variable zu erhalten.
3. Ein zweites Displacement wird zur Basisadresse addiert, um die endgültige Operandenadresse zu erhalten. (Bei externen Zugriffen kann man skalierte Indizierung benutzen, so daß man auch auf komplexe Datenstrukturen, z. B. Arrays, extern zugreifen kann.)

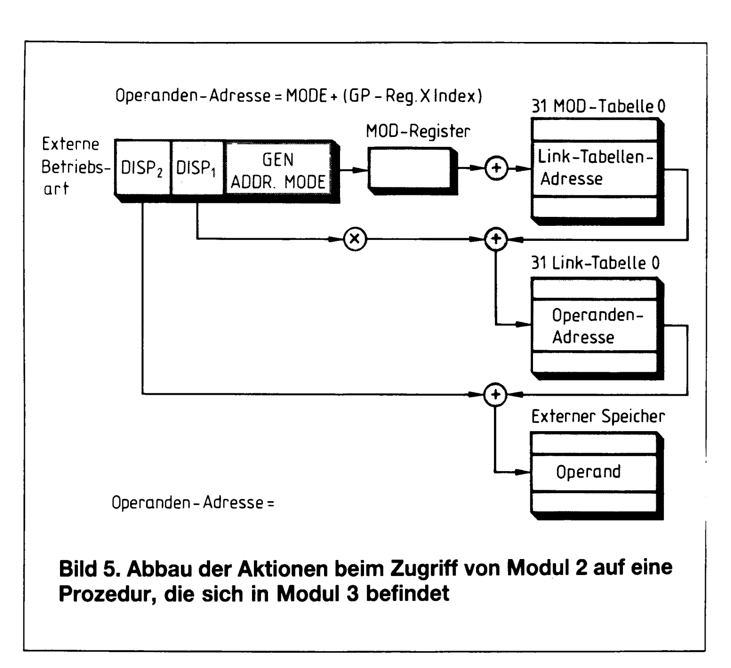


Bild 5. Abbau der Aktionen beim Zugriff von Modul 2 auf eine Prozedur, die sich in Modul 3 befindet

Wenn eine Prozedur, die sich in einem anderen Modul befindet, ausgeführt werden soll, ist der Ablauf etwas komplizierter, weil das SB-Register auch verändert wird, um die Zugriffe auf die statischen Daten externer Prozeduren zu beschleunigen. *Bild* 5 faßt die Sequenzen zusammen, die ablaufen, wenn Modul 2 eine Prozedur in Modul 3 aufruft. Wie bei externen Variablenzugriffen wird das im Befehl enthaltene Displacement als Offset für die Link-Tabelle von Modul 2 benutzt. Der derzeitige Status von Modul 2 wird gerettet, indem der Inhalt der MOD- und PC-Register in den Stack gebracht werden.

Als nächstes wird die Adresse des Deskriptors von Modul 3 von der Link-Tabelle des Moduls 2 gelesen und im MOD-Register abgelegt. Vom neuen Modul-Deskriptor wird die Adresse des statischen Datenblocks des Moduls 3 gelesen und im SB-Register gespeichert. Die Startadresse des Programm-Code-Blocks wird aus dem Moduldeskriptor gewonnen. Der Anfangspunkt innerhalb des Moduls, der aus der Link-Tabelle von Modul 2 stammt, wird zur gefundenen Programmbasis addiert, woraus sich der korrekte PC-Wert ergibt. Die CPU läuft jetzt in der Programmumgebung von Modul 3, so daß es möglich ist, weitere externe Prozeduren in anderen Modulen aufzurufen, falls das erforderlich ist.

Der komplementäre Befehl RXP (Return from external Procedure) bringt die Register SB, MOD und PC wieder in den Ursprungszustand zurück und entfernt aus dem Stack die Parameter, die an die externe Prozedur weitergegeben wurden. Die MOD- und PC-Werte werden vom Stack zurückgewonnen, wo sie durch die vorhergehende CXP-Instruktion abgelegt waren. Der SP-Wert wird vom Deskriptor des Moduls 2 gelesen.

Die hier beschriebenen modul-orientierten Mechanismen zeigen einen der wichtigsten Aspekte der Mikroprozessor-Weiterentwicklung. Ein Vergleich der Bauelemente, die während der letzten zehn Jahre auf den Markt gekommen sind, zeigt ein wachsendes Maß an Leistung, allerdings ist die Leistung allein nicht unbedingt ein absolutes Maß für die Verbesserung eines Mikroprozessors. Die optimierte Architektur der Serie 32000 unterstützt in vollem Maße das Entwickeln strukturierter Software, was heute wichtigster Teil aller Mikroprozessorapplikationen ist.

Günther Hausmann

Ein entscheidender Beitrag zur System-Leistungsfähigkeit:

Hochoptimierende Compiler für die 32000-Serie

Mit dem erwarteten Wachstum des Marktes für 32-Bit-Mikroprozessoren zu einem Milliarden-Dollar-Geschäft bis zum Ende der 80er Jahre kommt auch das Werben um die Kunden in eine neue, von härterer Konkurrenz gekennzeichnete Phase. Die Entscheidung des Kunden für einen bestimmten Prozessor basiert heute mehr auf der Überlegung, in welchem Maße dieser Baustein die Leistung eines Gesamtsystems erhöht, als auf einfacher Bewertung der reinen Prozessorleistung. In Anerkennung dieser wichtigen Tatsache hat National Semiconductor damit begonnen, das System-Software-Angebot zu erweitern und zu verbessern sowie ganz speziell eine eigene, fortschrittliche Compiler-Technik entwickelt.

Diese Technik, so meinen die dafür Verantwortlichen, wird die Kosten für Compiler senken helfen, eine einheitliche Portierbarkeit auf Unix, VMS und maschinennahe Betriebssysteme sicherstellen sowie – als wichtigstes Ergebnis – die Arbeitsgeschwindigkeit der 32000-Prozessoren erheblich steigern. Das Ergebnis der Investitionen von National Semiconductor ist die Entwicklung einer Anzahl von optimierenden Compilern für die Sprachen „C“, Fortran, Pascal und Modula II, die einen optimalen Code für die NS32000-Architektur generieren. Die neuen Compiler „Compiler Technology (CT)“ erzeugen Maschinencodes, die zwischen 30 und 200 % schneller ablaufen als bisherige. Sie leisten damit einen erheblichen Beitrag zur System-Leistungsfähigkeit der Serie 32000. *Tabelle 1* zeigt einen Vergleich der Ablaufzeiten verschiedener Benchmark-Programme mit einem herkömmlichen Compiler und einem neuen CT-Compiler.

PICTURE [fe],[fe],[fe]->[opt]->[cgen] HERE

Die CT-Compiler im Überblick

Die meisten Compiler der jetzigen Generation erzeugen den compilierten Code in zwei Durchläufen. Der erste Durchlauf, als „Front End“ bezeichnet, dient dem Lesen des Quellenprogramms, dem Prüfen der Syntax und Semantik und der Übersetzung in eine Zwischensprache („Intermediate Representation“ IR32). Im zweiten Ablauf, „Back End“ genannt, erfolgt die Übersetzung von der Zwischensprache in die jeweilige Maschinensprache. Bei den optimierenden CT-Compilern wird diese Technik noch erweitert, indem „Front End“ und „Back End“ vollständig voneinander getrennt werden und dazwischen ein Optimierungs-Ablauf liegt. Das „Front End“ wickelt alle Angelegenheiten ab, die mit der Quellensprache zu tun haben, wie Syntax, Zeichen-Definitionen sowie „Scope Rules“, und übersetzt fehlerfreie Programme in die Zwischensprache. Das „Back End“, oder auch der Code-Generator, ist auf die Prozessor-Charakteristik abgestimmt und übersetzt die Zwischensprache-Befehle in optimale Maschinencode-Sequenzen. Zwischen diesen beiden „steht“ der Optimierer, der das Zwischensprache-Programm liest, verfeinerte Code-Transformationen und Optimierungen erstellt und diese neutransformierte Version des Programmes in Zwischensprache ausgibt. Der Optimierungs-Durchlauf ist optionell und kann ausgelassen werden, um die Geschwindigkeit der Compilierung zu erhöhen.

Die einzigartige Zwischensprache der CT-Compiler, genannt IR32, bewirkt eine Trennung zwischen dem sprachbezogenen „Front End“ und den anderen Teilen des Compilers. Die IR32-Sprache enthält keine Elemente einer spezifischen Programmiersprache. Diese Unab-

hängigkeit des Interfaces von einer Sprache ist ein wesentlicher Vorteil: ein einziger Optimierer und ein Code-Generator unterstützen alle vier CT-Compiler. Die Entwicklung eines neuen optimierenden Compilers für eine zusätzliche Programmiersprache ist ebenso einfach wie das Schreiben eines neuen „Front Ends". Die Struktur der CT-Compiler zeigt *Bild 1*.

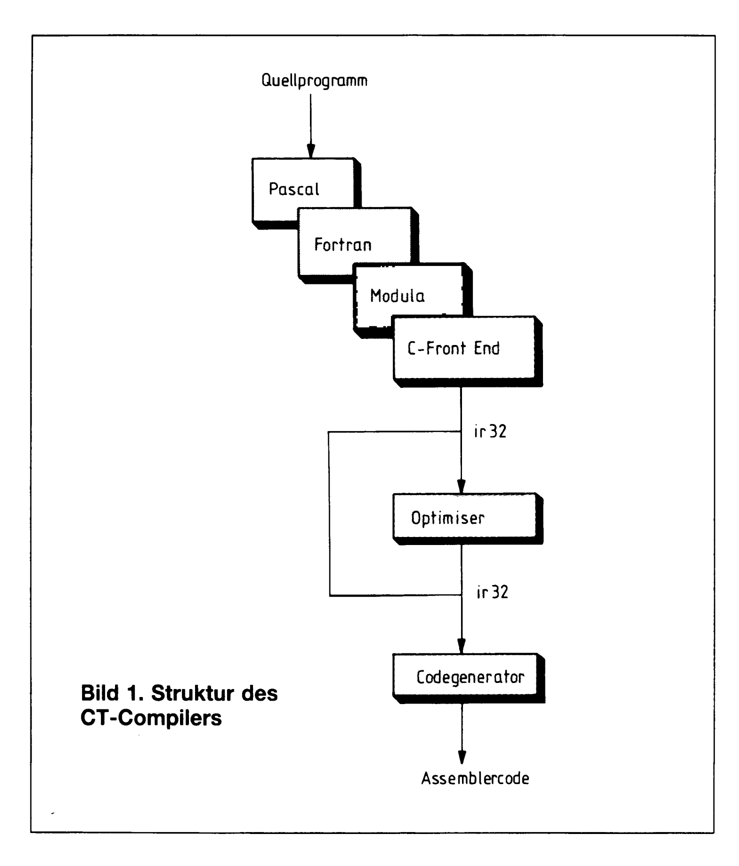


Bild 1. Struktur des CT-Compilers

Die IR32-Zwischensprache

Die Abkürzung „IR32" steht für „Intermediate Representation for the Serie 32000". IR32 beschreibt ein Programm in Form einer Sequenz von Aktionen und Anweisungen. Die Aktionen, in Form eines „Baumes", drücken den Programm-Algorithmus aus. Jeder „Zweig" des Baumes kann eine einfache Operation sowie eine Konstante oder Variable sein, der ein Satz von Attributen zugeordnet ist. Diese Attribute liefern die zusätzlichen Informationen über den Baum-Knoten, wie z. B. Typ und Format des Operanden. Alle Steuerstrukturen auf hoher Ebene, wie „if-then-else" und „while", sind in der IR32-Sprache als einfache oder bedingte „goto"-Befehle ausgedrückt.

Die IR32-Anweisungen liefern darüber hinaus Informationen wie Programm-Labels, Angaben über den Gültigkeitsbereich und Informationen für das Austesten aller Quellenprogramm-Ebene. *Bild 2* zeigt ein Beispiel für eine IR32-Aktion. Die IR32-Sprache ist eine architekturspezifische Umsetzung: die meisten der Operations-Knoten in den Ablaufbäumen entsprechen den Basis-Operatoren der NS32000-Prozessoren. Diese Eigenschaft führt zu einer präzisen Anpassung der CT-Compiler an die spezifischen Möglichkeiten der NS32000-Familie und führt zu einer optimalen Lösung. Das ist auch der Grund dafür, daß ein CT-generierter Code 5 bis 15 % schneller ist als der Code anderer Compiler, auch wenn der Optimierer nicht benutzt wird.

Die drei Compiler-Stufen

Front End

Da die Optimierer und Code-Generierungs-Back-Ends in allen CT-Compilern gleich sind, ist es das „Front End", das jeweils die Funktionen der verschiedenen Compiler bestimmt. Die CT-Compiler-Familie besteht aus Compilern für die vier meistverbreiteten technisch-wissenschaftlichen Programmiersprachen: „C", Fortran 77, ANSI-Pascal und Modula II. Die Erstellung von „Front Ends" für zusätzliche Sprachen, wie z. B. „Chill", wird in Betracht gezogen.

Der C-Compiler ist eine vollständige Implementation der Original-Definitionen von „C" mit den wichtigsten Zusätzen der bislang noch nicht veröffentlichten ANSI-Standard-Definition, einschließlich speziellen Merkmalen für Systemcode, der asynchrone Abläufe enthält. Darüber hinaus ist der C-Compiler vollständig kompatibel mit dem verbreiteten Unix „pcc"-C-Compiler.

Der Fortran-Compiler ist eine komplette ANSI-X3.9-1978-Standard-Implementierung mit allen Unix-f77-Erweiterungen.

Der Pascal-Compiler entspricht komplett ANSI/X3J9-1981, ISO-Ebene-1-Implementation, mit einigen wesentlichen Verbesserungen, wie z. B. modulare Compilierungs-Möglichkeiten.

Der Modula-II-Compiler ist eine originalgetreue Implementation der überarbeiteten Sprachendefinition von N. Wirth.

Alle Compiler unterstützen eine komplette Entwicklungsumgebung, einschließlich symbolischem Quellen-

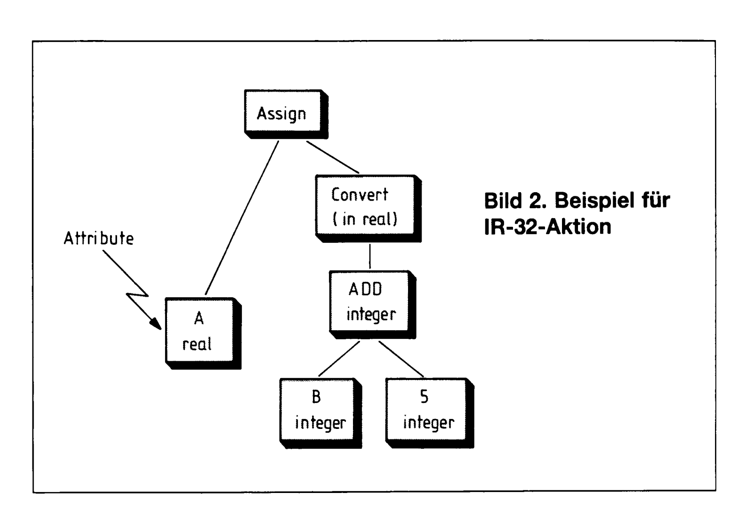


Bild 2. Beispiel für IR-32-Aktion

Debugger und Programm-Profiler. In allen Sprachen werden dieselben Aufruf-Konventionen benutzt. Prozeduren, die in einer Sprache geschrieben sind, können deshalb mit anderen verbunden werden, die in anderen Sprachen oder in Assembler geschrieben sind.

Die „C"- und Fortran-Compiler sind bereits verfügbar, Pascal- und Modula-Compiler werden noch 1986 zur Verfügung stehen.

Der Optimierer

Der CT-Optimierer, der ein Kernstück aller CT-Compiler darstellt, basiert auf einer fortschrittlichen Optimierungs-Theorie, die in den letzten 15 Jahren entwickelt wurde. Hauptelement ist eine innovative Datenfluß-Analyse-Technik, die eine einfache Implementierung des Optimierers ermöglicht. Hierdurch werden einzigartige Optimierungen zusätzlich zu den in anderen Compilern anzutreffenden Standard-Optimierungen möglich. Die Optimierung erfolgt global innerhalb des Codes einer gesamten Prozedur und nicht nur in einem lokalen Zusammenhang. Die Optimierung läßt sich als Mehrschritt-Prozeß betrachten. Jeder Schritt erzeugt eine spezielle Optimierung und die Voraussetzung für neue Optimierungsmöglichkeiten innerhalb des nächsten Schrittes.

Der erste Schritt im gesamten Optimierungsprozeß ist das Lesen des IR32-Programms in einem zeitlichen Durchlauf und die Aufteilung dieser Gesamt-Prozedur in „Basic-Blocks". Ein Basis-Block ist eine aufeinanderfolgende Code-Sequenz, die nur am Anfang oder am Ende Verzweigungen hat. Einige der lokalen Optimierungen in der ersten Stufe sind:

- Wert-Zuweisung (Value Propagation)
  Value Propagation ist der Versuch, eine Variable durch den aktuellen Wert zu ersetzen, der dieser Variablen zugeordnet worden ist. Diese Optimierung, meist als „Copy Propagation" bezeichnet, ist deshalb wichtig, weil sie neue Möglichkeiten für weitere Optimierungen schafft, siehe dazu auch das Beispiel in *Bild 3*.
- Konstanten-Zusammenfassung (Constant Folding)
  Falls ein Ausdruck oder eine Bedingung nur aus Konstanten besteht, bildet der CT-Optimizer daraus eine Konstante, damit dieser Rechenvorgang nicht während des Programmablaufs stattfinden muß. Der Optimizer bildet mitunter auch Ausdrücke um, so daß ein „Constant Folding" bei Teilen von Ausdrücken möglich wird, in denen algebraische Funktionen wie das Kommutativ-, Assoziativ- und Distributiv-Gesetz Verwendung finden.
- Eliminierung redundanter Zuweisungen
  Der Optimizer erfaßt und eliminiert Zuweisungen zu Variablen, die später im Programm nicht mehr gebraucht werden oder neu zugewiesen werden müssen, bevor sie wieder benutzt werden.

Die Programmsequenz
```
a := 4:
if a*8 < 0 then
    b := 15:
else
    b := 20:
... code which uses b ...
```
wird vom Pascal-Frontend in folgenden Zwischencode umgesetzt:
```
    a = 4
    if (a*8 >= 0) goto L1
    b = 15
    goto L2
L1: b = 20
L2: ...
```
Dies wird durch Werteweitergabe umgesetzt in:
```
    a = 4
    if (4*8 >= 0) goto L1
    b = 15
    goto L2
L1: b = 20
L2: ...
```
Das wird durch Konstantenfaltung umgesetzt in:
```
    a = 4
    if (true) goto L1
    b = 15
    goto L2
L1: b = 20
L2: ...
```
Entfernung des „Toten Codes" führt zu:
```
    a = 4
    goto L1
L1: b = 20
L2: ...
```
Durch eine einfache Flußoptimierung ergibt sich:
```
    a = 4
    b = 20
    ...
```
Weil es keine weitere Verwendung für redundante Zuweisungen gibt, reduziert sich der Code auf:
```
    b = 20
    ...
```
Bild 3. Bezeichnung zwischen verschiedenen Optimierungen

Der zweite Schritt im Optimierungs-Prozeß ist der Aufbau des Programm-Ablaufdiagramms. In diesem Diagramm steht jeder Knotenpunkt für einen Basis-Block. Von einem zum anderen Knotenpunkt (Basis-Block) wird ein Pfeil gezeichnet, wenn ein logischer Pfad in der Prozedur vorhanden ist, der vom ersten Basis-Block zum zweiten führt. Während des Aufbaus des Programm-Ablaufdiagramms können weitere Optimierungen erfolgen:

- Ablauf-Optimierungen
  Der Zweck von Ablauf-Optimierungen ist die Reduzierung der Zahl von Verzweigungen im Programm. Ein Beispiel ist das Ersetzen einer Verzweigung, deren Ziel eine andere Verzweigung ist, durch eine direkte Verzweigung zum endgültigen Ziel. In den anderen Fällen wird der Code so verändert, daß die überflüssigen Verzweigungen eliminiert werden.
- Eliminierung von nicht auszuführendem Code
  Die Ablauf-Optimierung ist darauf ausgelegt, dem Optimierer zu helfen, den Code herauszufinden, der

nicht aktuell abgearbeitet wird. Die Eliminierung dieses Codes, „Dead Code Elimination" genannt, führt zu kürzeren Objekt-Programmen.
Bild 3 zeigt die Beziehung zwischen einigen Optimierungen.

Schritt drei des Optimierungs-Prozesses ist die globale Datenfluß-Analyse (Global Data Flow Analysis). Ziel dieser Analyse ist es, globale Code-Transformationen herauszufinden, die den Programmablauf beschleunigen. Viele Analysen konzentrieren sich auf die Beschleunigung von Schleifenabläufen, da die meisten Programme 90 % und mehr der Ablaufzeit in Schleifen laufen. Die globale Datenfluß-Analyse erfordert die Berechnung einer großen Zahl von Merkmalen für jeden Ausdruck innerhalb der Prozedur.

Im Gegensatz zu den meisten Optimierern, die einzelne nicht miteinander verbundene Techniken nutzen, ist der CT-Optimierer um eine innovative Technik herum aufgebaut, die die Erfassung der Programmsituation erlaubt und „Partial Redundancy" genannt wird. Diese Technik ist so leistungsfähig, daß die meisten Optimierer als ein Spezialfall dieses Optimierers angesehen werden können. Dahinter steht der wesentliche Gedanke, daß es ineffizient ist, einen Ausdruck, z. B. a×b, zweimal auf demselben Pfad zu berechnen. Es geht oft schneller, das Ergebnis der ersten Berechnung zu speichern und dann die redundante zweite Berechnung durch den gespeicherten Wert zu ersetzen. Noch weiter verbreitet ist jedoch der Fall, in dem ein Ausdruck partiell redundant ist. Das heißt es existiert ein Pfad, der zu einem Ausdruck führt. Dieser Pfad enthält die entsprechende Berechnung. Es existiert noch ein zweiter Pfad ohne Berechnung.

Der folgende Code enthält einen partiell redundanten Ausdruck (a*b):

```
if C then
    x := a*b
else
    b := b+10:
x := a*b:
```

Zunächst erfolgt Umsetzung in einen vollständig redundanten Ausdruck:

```
if C then
    x := a*b
else
    b := b+10:
    temp := a*b:
y := a*b:
```

Danach läßt sich der Code folgendermaßen reduzieren:

```
if C then
    temp := a*b
    x := temp
else
    b := b+10:
    temp := a*b:
y := temp:
```

Jetzt wird der Ausdruck „a*b" nur einmal pro Durchlauf be-

Bild 4. Beispiel für Eliminierung partielles Redundanzen

Die folgenden Optimierungen werden mit der herkömmlichen Technik durchgeführt:

- Eliminierung von vollständig redundanten Ausdrücken
  Diese Optimierung wird oft als „Common Subexpression Elimination" bezeichnet. Es ist relativ einfach, vollständig redundante Ausdrücke zu eliminieren. Der Optimierer speichert dabei das Ergebnis der ersten Berechnung (gewöhnlich in einem Variablen-Register) und benutzt den gespeicherten Wert, anstatt die zweite Berechnung durchzuführen. Programmierer, die gewissenhaft auf Leistung programmieren, berücksichtigen dies von selbst. Aber in vielen Fällen, z. B. bei der Array-Index-Berechnung, kann nur der Optimierer weiterhelfen.
- Partiell-Redundanz-Eliminierung
  Ein partiell redundanter Ausdruck ist in zwei Schritten eliminierbar. Zuerst wird der Ausdruck in die Pfade eingespeist, in denen er bislang noch nicht aufgetreten war. Dies macht den Ausdruck vollständig redundant. Im Schritt zwei wird die erste Berechnung gespeichert und es werden die redundanten Berechnungen durch den gespeicherten Wert ersetzt. Ein Beispiel für diese Optimierung ist in *Bild 4* zu sehen. Diese Optimierung führt mitunter zu etwas längeren Codes, die Ausführung wird dadurch aber nicht beeinträchtigt, weil alle eingefügten Ausdrücke parallel anstehen und nur einer jeweils berechnet wird.
- Veränderung von Ausdrücken, die von Schleifendurchlauf unabhängig sind (Loop Invariant Code Motion)
  Wenn ein Ausdruck in einer Schleife auftritt und der Wert dieses Ausdrucks sich beim Durchlaufen der Schleife nicht verändert, so bezeichnet man dies als „Loop Invariant". Loop-Invariant-Ausdrücke sind ebenfalls partiell redundant. Dies wird verständlich, wenn man bedenkt, daß es zwei Pfade in die Schleife gibt. Einer führt durch den Schleifen-Anfang (wenn die Schleife zum ersten Mal ausgeführt wird) und der andere führt durch das Ende der Schleife, wenn die Exit-Bedingung falsch ist. Loop-Invariant-Berechnungen werden folglich auf dieselbe Weise aus der Schleife entfernt: der Ausdruck wird zuerst in den Eingangspfad der Schleife gelegt, dann in einem Register gespeichert, während die redundante Berechnung in der Schleife durch den Register-Inhalt ersetzt wird.

Der vierte Optimierungsschritt des CT-Optimierers – und möglicherweise der profitabelste – ist die Register-Zuordnungs-Phase (Register Allocation Phase). Register Allocation ist der Prozeß der Zuordnung von Variablen zu Maschinen-Registern anstatt Hauptspeicher-Plätzen. Register-Zuweisungen sind immer viel schneller und benötigen weniger Codelänge als vergleichbare Hauptspeicher-Zuweisung.

Der Algorithmus, der vom Optimierer benutzt wird, heißt „Coloring Algorithm". Die erste globale Ablauf-Analyse wird dabei ausgeführt, um die verschiedenen Lebensdauern („Live Ranges") der Variablen innerhalb der Prozedur zu ermitteln. Die Lebensdauer entspricht

dem Stück eines Programmpfades, bei dem eine Variable einen bestimmten Wert hat. Generell beginnt mit jeder Zuweisung zu einer Variablen eine neue Lebensdauer. Diese endet mit dem jeweils letzten Gebrauch des zugewiesenen Wertes.

Der Optimierer erzeugt im nachhinein auf folgende Weise einen Graphen: Jeder Knotenpunkt im Programm stellt eine „Live Range" dar. Zwei Knotenpunkte werden dann verbunden, wenn ein Punkt im Programm existiert, an dem zwei Lebensdauern sich überschneiden. Die Zuordnung der Register zu den Lebensdauern ist nun dieselbe wie beim „Coloring" der Knotenpunkte des Graphen, so daß zwei verbundene Knotenpunkte zwei verschiedene „Colors" haben. Dies ist ein klassisches Problem der Graphen-Theorie, für das es gute Lösungen gibt. Falls nicht genügend Register vorhanden sind, so haben öfter verwendete Variable eine höhere Priorität als weniger oft gebrauchte. Die Verschachtelung von Schleifen wird bei der Ermittlung der Häufigkeit des Gebrauchs einer Variablen berücksichtigt. Das bedeutet, Variable, die innerhalb von Schleifen benutzt werden, haben eine höhere Priorität als andere.

Die meisten optimierenden Compiler versuchen eine Registerzuordnung nur für echte lokale Variable, bei denen nicht die Gefahr eines „Aliasing" besteht. Ein „Alias" tritt auf, wenn es zwei unterschiedliche Möglichkeiten gibt, um auf eine Variable zuzugreifen. Dies kann dann passieren, wenn eine globale Variable als Referenz-Parameter weitergeleitet wurde. Auf diese Variable kann dann über ihren Globalnamen oder über das entsprechende „Alias"-Parameter zugegriffen werden. Derselbe Fall tritt in „C" auf, wenn die Adresse einer Variablen einem Zeiger zugeordnet wurde.

Bei dem CT-Optimierer wurde ein umfassender Ansatz gewählt, um alle Variablen mit deren jeweiligen Datentypen für die Registerzuordnung zu berücksichtigen, einschließlich globaler Variablen, Variablen deren Adressen berechnet werden, Array-Elemente und Ausdrücke, die durch einen Zeiger bestimmt werden. Diese speziellen Variablen können nicht über Prozeduraufrufe oder Zeigerreferenzen in Register gebracht werden und haben deshalb im allgemeinen eine geringere Priorität als lokale Variable. Damit sie jedoch nicht völlig „disqualifiziert" werden, fällt die Prioritätsentscheidung jeweils durch den „Color"-Algorithmus.

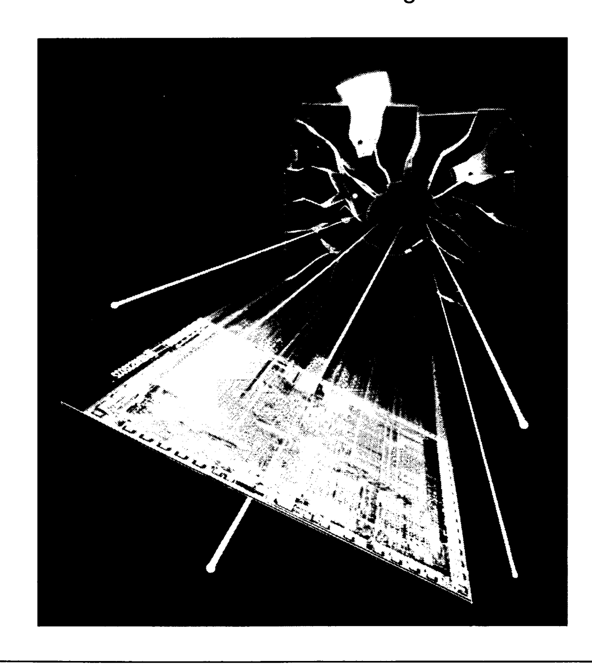
Zusätzliche, wichtige Optimierungen, die durch die Register-Zuordnung entstehen, sind:

– Gebrauch von flüchtigen Registern
  Die Register der NS32000-Prozessoren sind durch eine Festlegung in zwei Gruppen geteilt: solche, die am Prozedur-Anfang/-Ende gesichert/aufgefrischt werden und andererseits die sogenannten flüchtigen Register, die als temporäre Speicher ohne Sicherung des Inhaltes arbeiten. Der Register-Zuordner sorgt für einen maximalen Einsatz der flüchtigen Register, um damit den Overhead der Prozeduraufrufe zu reduzieren.
– Register-Parameter-Zuordnung
  Die Register-Zuordnung versucht Routinen zu erfassen, deren Parameter in die Register geleitet werden

Die Compiler auf einen Blick

Die wesentlichen Eigenschaften der zukunftsweisenden Compiler-Technologie von National Semiconductor:

- Der C-Compiler ist eine vollständige Kernighan-und-Ritchie-Implementierung, kompatibel mit den meisten gebräuchlichen C-Compilern.
- Der Fortran-Compiler entspricht vollständig den Anforderungen nach ANSI-X3.9-1978 und enthält einige Erweiterungen sowie mehrere bekannte Preprozessoren.
- Der Pascal-Compiler ist eine vollständige Implementierung nach ANSI/X3J9-1981 mit einigen Erweiterungen wie modulare Programmierung.
- Alle Compiler sind in C geschrieben und damit leicht auf verschiedene Umgebungen zu portieren.
- Alle Compiler erzeugen extrem effizienten und dichten Code.
- Alle Compiler arbeiten mit dem gleichen Hochleistungsoptimierer und dem gleichen Codegenerator. Der erzeugte Code ist durchgängig von guter Qualität.
- Alle Compiler benutzen die gleichen „Calling"-Vereinbarungen. Prozeduren aus der einen Sprache können deshalb mit Prozeduren aus einer anderen Sprache oder mit dem Assembler gemischt zusammenarbeiten.
- Alle Compiler unterstützen die gesamte 32000-Familie.
- C- und Modula-Compiler haben spezielle Schalter für Systemcode, der asynchrone Ereignisse enthält.
- Der gemeinsame Codegenerator ist hochoptimiert und kann wahlweise die Ausführungsgeschwindigkeit oder die Codedichte optimieren.
- Alle Compiler sind als „native" Compiler für GENIX (bsd4.1) und UNIX-System V erhältlich oder als Cross-Compiler für VAX/VMS und VAX/UNIX (bds4.2).
- Es steht eine vollständige umfangreiche Entwicklungsumgebung zur Verfügung wie Profiler, Assembler, Source-Level-Debugger für lokale Hardware und Zielhardware sowie für National Real-Time-In-System-Emulatoren.
- Die Eigenschaften der neuen Compiler-Technologie werden von allen CPUs der Serie 32000 genutzt.



können. Dies ist nur für interne Routinen möglich, da alle Aufrufe solcher Routinen für den Optimierer erkennbar sind. Aufrufe für andere externe Routinen sind Bestandteil der NS32000-Standard-Aufruf-Sequenz. Die Weiterleitung von Parametern in Registern ist ein anderer Weg zur Reduzierung des Overheads von Prozedur-Aufrufen.
Der letzte Optimierungsschritt sichert die Ergebnisse aller vorherigen Schritte durch Schreiben der optimierten Prozedur in IR32-Form. Einige Reorganisationen finden auch während dieses Schrittes statt. Lokale Variable, die den Registern zugeordnet sind, werden aus dem Aktivierungsteil der Prozedur entfernt. Anschließend wird dieser Teil neuorganisiert, um den Gesamtumfang zu minimieren.

Das „Back End"

Das CT-Back-End (Code-Generator) hat die Aufgabe, für die IR32-Ausdrücke optimale Code-Sequenzen zu bilden. Zur Minimierung des Gebrauchs von temporären Registern, die für die Berechnung von Unterausdrücken im Programmablauf notwendig sind, werden Standard-Techniken eingesetzt. Die Hauptstärke des Code-Generators liegt in der Zahl von „Peephole"-Optimierungen, die er erzeugt. „Peephole"-Optimierungen sind maschinenabhängige Code-Transformationen, die vom Code-Generator in Form von kleinen Maschinencode-Sequenzen vor der Code-Ausgabe erzeugt wurden. Einige der wichtigsten „Peephole"-Transformationen sind:

- Entfernen des Codes zur Verwaltung des „Frame of Routines", die keine lokalen Variablen haben oder bei denen alle Variablen bestimmten Registern zugeordnet sind.
- Ständiger Abgleich der Stack- und Frame-Bereiche zur Minimierung der Datenzugriffe.
- Reduzierung identischer arithmetischer Ausdrücke wie $X \times 1 = X$, $X + 0 = X$ usw.
- Einsatz von ADDR-Anweisungen anstatt ADD mit drei Operanden.
- Verwendung von ADDR anstatt MOVZBD mit kleinen Konstanten.
- Einsatz von MOVD anstatt MOVF sowie Index-Adressierungs-Modus anstatt Multiplikationen mit 2, 4 oder 8; Kombinationen von ADDR-Anweisungen anstatt Multiplikationen mit anderen Konstanten bis 200.
- Einige Optimierungen, wie zum Beispiel die Verwendung des Distributiv-Gesetzes der Algebra (z. B.: $(10 + i) \times 4 = 40 + 4 \times i$), eröffnen zusätzliche Möglichkeiten für den Code-Generator zur vollen Ausnutzung der Adressierungs-Modi bei der 32000-Familie.
- Optional kann der Optimierer auf eine Betriebsart umgeschaltet werden, die stärker auf die Erzeugung eines kompakten Codes ausgelegt ist als auf die Erzeugung eines möglichst schnellen Codes.

Zusammenfassung

Die Geschwindigkeit eines Mikroprozessors ist nicht mehr nur eine Funktion der Anzahl seiner Gatter oder des Herstellungsprozesses. Heute erwarten die Anwender eine vollständige Hochleistungs-Systemumgebung. Die optimierenden CT-Compiler von National Semiconductor wurden unter Nutzung der neuesten Erkenntnisse der Compiler-Technik entwickelt und nutzen die Möglichkeiten der 32000-Architektur voll aus. Anwendungen auf Basis der 32000-Familie erreichen dank dieser neuen Compiler eine völlig neue Leistungsebene.

Hermann Hösl

Applikationen unter Unix

Als die amerikanische AT&T Corporation zu Beginn des Jahres 1984 in einer groß angelegten Marketing-Kampagne („Consider it standard") begann, Unix-System V kommerziell zu verwerten, herrschte allerorten Skepsis über dieses Vorhaben. Inzwischen ist einige Zeit vergangen und System V ist zum De-facto-Standard in der Unix-Familie geworden. Auch Microsoft hat sich mit Xenix-5.0 auf die Kompatibilität zu System-V festgelegt, ebenso wie Digital Equipment mit seinem „Ultrix" – eine Version von Berkeley 4.2. Darüber hinaus haben sich mit Intel, Motorola, National Semiconductor und Zilog auch schon die großen Chip-Hersteller zu System V bekannt. Kein Hersteller kann es sich leisten, ohne Unix auszukommen.

Unix auf dem Vormarsch

Unix läuft auf den Rechnern von über 60 großen Herstellern, wobei die Palette von den PCs bis zu Mainframes reicht: der Anwendungsschwerpunkt für Unix liegt, und daran wird sich auch in Zukunft nichts ändern, bei den Micros und Minis. Daß über Unix nicht nur viel geschrieben wird, beweist die Zahl von Installationen, die von (geschätzten) 300 000 in 1985 auf vielleicht über eine Million in 1986 steigt. Den Löwenanteil hat dabei die Microsoft-Linie mit Xenix, weit vor der AT&T-Familie mit Version 7, System III und System V der Berkeley-Linie mit 4.1-, 4.2- und 4.3-BSD sowie den zahlreichen Look-Alikes und Work-Alikes. Daß bei dieser verwirrenden „Artenvielfalt" die Rufe nach einem Standard immer lauter wurden, verwundert nicht. Schon 1981 formierte sich in der „/usr/group" (eine kommerziell orientierte Vereinigung mit dem Ziel der Vermarktung von Unix) das „Standard Committee" zur Formulierung und Publizierung eines Betriebssystem-Standards in der kommerziellen Unix-Welt. Das Komitee, besetzt mit Herstellern von Unix-Systemen, Unix-ähnlichen Systemen und Benutzern, brachte Ende 1984 einen Vorschlag zur Vereinheitlichung der Unix-System-Calls und Subroutine-Libraries heraus. Noch weiter geht die Arbeit der „Open Unix Group X/OPEN", die im August 1985 einen europäischen Standard, basierend auf der System-V-Interface-Definition von AT&T und dem „/usr/group"-Vorschlag, erarbeitet hat. Die Gruppe, die sich nach den Initialen ihrer Mitglieder (Bull, ICL, Siemens, Olivetti und Nixdorf) zuerst „BISON" nannte, und zu der später noch Philips stieß, hat mit ihrem „Portability Guide" einen wesentlichen Beitrag sowohl zur Standardisierung von Unix-System-V als auch der darauf ablaufenden Software geliefert. Wesentliche Bestandteile des Standardisierungswerkes sind:
- System-V-Spezifikation (XVS),
- C-, COBOL- und Fortran-Standards,
- Indexed-Sequential-Access-Method (ISAM),
- Source-Code-Transfer, eine Definition von Hardware- (Floppy, Tape) und Software-Standards (tar, cpio, uucp)

für die Übertragung zwischen Unix-Maschinen.

Geplant sind für die Zukunft die Definition von Standards für
- Netzwerkverbindungen,
- Inter-Prozeß-Kommunikation,
- verteilte Filesysteme,
- Filesysteme,
- Transaction-Processing,
- Datenbank-Schnittstelle (SQL),
- Curses-Library,
- Grafik (GKS),
- Pascal-Sprachdefinition.

Von dieser „Unix-Bibel" erwartet man sich auf der einen Seite Vorteile für die „Independent Software Vendors" (ISV), also die herstellerunabhängigen Software-Produzenten, denen sich mit dem Standard ein weiter Markt erschließt. Zum anderen sind Benutzer nicht mehr von einem Hersteller abhängig und können Applikations-Software leicht von einem System auf ein anderes transportieren. Damit hängt die Portierbarkeit von Applikations-Software – dem eigentlichen Problem in der ganzen Diskussion um Unix-Portabilität – wirklich nur noch von der Disziplin der Programmierer ab.

Unix als Entwicklungswerkzeug

Will man Applikationen unter Unix beschreiben, so muß man mit der Anwendung beginnen, für die es ursprünglich geschrieben und kontinuierlich fortentwickelt wurde, nämlich der Programmentwicklung. Unix vereinigt mit über 200 Utilities eine große Menge von Bausteinen, die auf einfachste Weise zu immer neuen Anwendungen zusammengefügt werden können. Der „Kitt", der diese Bausteine zusammenhält, ist die „Shell", die als Benutzerinterface an ihrer Mächtigkeit durchaus mit einer Programmiersprache vergleichbar ist. Drei Elemente von Unix, die dieses Baukastenprinzip erheblich unterstützen, sind „Filter", „Pipes" und eine einheitliche „Geräteschnittstelle". Als Filter

bezeichnet man solche Programme, die über den Standard-Eingang (Standard Interrupt) Eingaben empfangen, die Daten entsprechend verarbeiten und auf dem Standard-Ausgang (Standard Output) wieder ausgeben. Beide Schnittstellen sind, wie auch die dritte (Standard Error), normalerweise dem Benutzerterminal zugeordnet, können jedoch durch die einheitliche Geräteschnittstelle leicht einem anderen Gerät oder einer Datei zugeordnet werden. Mit Pipe schließlich bezeichnet man die Möglichkeit, die Ausgabe eines Programms in die Eingabe eines anderen überzuführen. Diese Techniken lassen sich leicht an einem Beispiel veranschaulichen:

a) ls
   Directory-Listing auf dem Bildschirm
b) ls | pr −2 −t
   Directory-Listing zweispaltig auf dem Bildschirm
c) ls | lp
   Ausgabe auf dem Drucker mittels Spooler
d) ls | pr −2 −t | lpr
   b) und c) zusammen
e) ls > file
   Ausgabe in ein Plattenfile
f) ls > /dev/null
   Ausgabe auf dem (Pseudo-)Gerät „Null"

Die Multi-Tasking-Merkmale von Unix erlauben dem Programmierer eine wesentlich höhere Produktivität in der Software-Entwicklung. Ein Unix-System ist durch jeden einigermaßen geschickten Programmierer leicht erweiterbar, sei es durch Shell-Scripts oder Programme, die in einer höheren Programmiersprache geschrieben sind. Dabei eignen sich Shell-Prozeduren vor allem für die Erledigung von kleineren Alltagsarbeiten, bei größeren Projekten sollte man auf C-Programme zurückgreifen. Unix beinhaltet etliche Utilities, für die man in anderen Systemen extra bezahlen müßte: Das Software-Verwaltungssystem „SCCS", das Programm „Make" zur automatischen Überwachung und Ausführung von Programmgenerierungsabläufen, „LEX" und „YACC", die Tools für den Schreiber von Compilern, „NROFF" und „TROFF" für die Textverarbeitung und schließlich die Bibliothek „Curses" für Programme, die die Fenstertechnik benutzen.

Allein mit den Standard-Tools SCCS, Make, Electronic Mail und AT (Start von Programmen zu einem gewissen Zeitpunkt) ließe sich ein einigermaßen komfortables Projektverfolgungssystem erzeugen, das die automatische Generierung von Objekten übernimmt und Mitarbeiter des Projektteams an überfällige Module erinnert. Wie schon gesagt: Unix liefert die Zutaten, sein „Süppchen" muß sich jeder selbst kochen. Unix unterstützt natürlich alle gängigen Programmiersprachen wie C, Pascal, Fortran, Cobol und Basic. Daneben existieren auch Compiler für spezielle Anwendungen in Programmiersprachen wie Lisp, Prolog, Forth und Modula-2. Auch von Ada, der vom „DoD" (Department of Defense = US-Verteidigungsministerium) als Standard eingeführten Sprache, gibt es bereits Implementierungen unter Unix.

Der Trend zu Unix

Unix hat sich von den Minis in beide Richtungen des Leistungsspektrums bewegt und seine eigentliche „Heimat" bei den Micros und Supermicros gefunden. Hier wird es seiner Rolle als De-facto-Standard gerecht. Auf dem Personal-Computer-Markt wird es Unix schwerer haben, sich gegen MS-DOS und PC-DOS durchzusetzen, was auch entscheidend von der weiteren Entwicklung der Applikations-Software unter Unix abhängt. Mit dem ständigen Anwachsen der Rechnerleistung und dem Trend hin zu 32-Bit-Prozessoren wird sich das Gewicht jedoch mehr und mehr zu Unix verlagern. Ein wichtiger Faktor ist auch die Halbleiter-Industrie, die, wie National Semiconductor, ein klares Bekenntnis zu Unix abgelegt hat. „Wir glauben nicht an die langfristige Zukunft von MS-DOS und PC-DOS", meint z. B. **Werner Trattnig**, Product-Marketing-Director bei National Semiconductor. Durch AT&T ist eine langfristige Weiterentwicklung und Unterstützung von Unix gewährleistet. Die einzige reale Basis für Multi-User-Micros ist Unix

Applikationen unter Unix

Es hat immer wieder Kritik an der Qualität von Anwendungs-Software unter Unix gegeben. Das mag seinen Grund darin haben, daß Unix lange Zeit ein System für Insider gewesen ist und die Zahl der Applikationspakete dementsprechend mager war. Verglichen mit Software-Paketen aus der IBM-PC-Welt müssen die Hersteller im Unix-Bereich sicher noch einiges aufholen, was Darstellung und Benutzerfreundlichkeit ihres Produktes betrifft. So machen Applikationspakete unter Unix häufig nicht allzuviel aus dem Terminal, auf dem sie ablaufen. Gerade hier ist der Benutzer aus der PC-Welt schon Besseres gewohnt. Dies wird sich in Zukunft, wenn die Zahl der Applikationen unter Unix steigt, ändern.

Man hat oft behauptet, daß Unix schlecht für kommerzielle Anwendungen sei. Im Prinzip zwar falsch, enthält diese Aussage doch ein Körnchen Wahrheit: so verdienen z. B. die Engpässe „Terminal-I/O" und „Disk-I/O" beim Entwurf eines Unix-Systems höchste Beachtung; Record- und File-Locking sind erst in der jüngsten Release von System V enthalten und die Programm-Ladezeit besonders auf Systemen ohne „Demand-Paging" macht sich recht störend bemerkbar.

Unix als Multi-User-, Multi-Tasking-, Time-Sharing-System hat Antwortzeiten, die durch gleichzeitig laufende Prozesse verschieden lang sein können; dies kann für Benutzer, die vom Single-User-PC kommen, geradezu schockierend wirken. Das alles heißt nicht, daß es nicht möglich ist, schnelle und effiziente Applikationen auf Unix-Maschinen zu implementieren, was die steigende Zahl von Paketen gerade im kommerziellen Bereich beweist.

Die Aussage, daß die Bedieneroberfläche von Unix eine Katastrophe für den ungeübten Benutzer sei, wird durch ihre häufige Wiederholung auch nicht wahrer. Für den Entwickler stellt Unix eine nahezu ideale Umgebung dar und für den nichtprogrammierenden Anwender existieren komfortable Menü-Shells, Fenstertechniken usw., die ihn mit dem eigentlichen Unix gar nicht in Berührung kommen lassen. Insofern ist es dem Benutzer von Anwendungsprogrammen relativ gleichgültig, auf welchen Betriebssystemen seine Applikationen laufen.

Es sind heute vor allem drei Anwendungsgebiete, die zunehmend auf Unix implementiert werden:
- CAD/CAM, CAE, CASE,
- Datenbanksysteme,
- Büroautomatisierung und kommerzielle Anwendungen.

Daneben gewinnt Unix immer mehr als Trägersystem für AI- und Expertensysteme an Bedeutung. Im CAD/CAM/ CAE/CASE-Bereich ist Unix zum De-facto-Standard geworden. Bei einem Vergleich von Personal-Workstations in der Zeitschrift „Systems International" (11/85) arbeiten von 25 Grafikstationen 19 auf der Grundlage von Unix oder einem Derivat. Dabei zeigt sich eine eindeutige Präferenz von Berkeley-4.2, die wohl darauf zurückzuführen ist, daß die Berkeley-Linie als erste die für Grafik-Applikationen unerläßliche virtuelle Speicherverwaltung aufwies. Die Hersteller bieten meist neben eigenen Produkten aus den verschiedenen Grafikbereichen auch Produkte von Third-Party-Firmen an.

InterPro-32 von Intergraph

Mit seiner multi-funktionalen Workstation InterPro-32, basierend auf dem Prozessor 32032 von National Semiconductor, bietet Intergraph vier Operationsmodi in einer Maschine an, von denen drei unter Unix ablaufen:
- Als „offenes System" unter Unix System V hat der „InterPro-32" die Möglichkeit, neben der Intergraph-CAE-Software auch selbstentwickelte Programme und Pakete von unabhängigen Herstellern laufen zu lassen. Der InterPro-32 ist eine vollständige Entwicklungsumgebung mit C- und Fortran-77-Compilern, Assembler, einer Workstation-Display-Library, Window-Environ-

Die Entwicklungsgeschichte von Unix

Die Geschichte von Unix fing in der Abteilung für Computer-Science-Research der Bell-Laboratories in Murray Hill, New Jersey, an, als das Ergebnis eines Gemeinschaftsprojektes zwischen Bell-Labs, General Electric und dem MIT. Ziel dieses Projektes war es, ein interaktives Multi-User-Betriebssystem zu entwickeln. Das Resultat: ein auf dem General-Electric-645-Großrechner laufendes Betriebssystem mit dem Namen „Multics". Obwohl Multics stetig weiterentwickelt wurde und auch heute noch auf Anlagen von Honeywell verfügbar ist, war man bei Bell mit dem Produkt wenig zufrieden und zog sich im März 1969 aus dem Projekt zurück. Der GE-645-Rechner wurde abgebaut, was dazu führte, daß die Gruppe sich einen neuen Computer suchen mußte und auf das Thema Betriebssysteme allgemein schlecht zu sprechen war.

Ken Thompson, der im selben Büro wie die Leute von der Computer-Science-Research-Gruppe saß, war vor allem an einem Betriebssystem für die Programmentwicklung interessiert. Speziell erwartete er sich Vorteile von der Entwicklung eines hierarchischen Filesystems. Thompson und zwei Kollegen, **Rudd Canaday** und **Dennis Ritchie**, entwarfen das Konzept für ein Filesystem, das Thompson dann auf einem Gecos-System simulierte.

Das Filesystem hatte die Struktur eines gerichteten Graphen: die „root"-Directory enthielt Files und Directories, die ihrerseits Datenfiles und Directories enthalten konnte. Jeder File wurde als eine direkt adressierbare Bytesequenz ohne besondere Eigenschaften betrachtet. Es war weder notwendig noch möglich, Platz für einen File zu reservieren. **Brian Kernighan**, der ebenfalls an dem Projekt mitarbeitete, hat diese simple Art, einen File darzustellen, vom Multics-I/O-System übernommen und somit das erste wirkliche Unix-Merkmal auf dem Gecos-System.

Neben ihrer Arbeit an dem Filesystem schrieben Thompson und Ritchie ein Programm mit dem Namen „Space Travel" für die Gecos-Maschine. Das Programm lief sehr langsam unter dem Gecos-Timesharing-System, so daß man sich entschloß, eine ausrangierte Single-User-PDP-7 mit einem 340-Bildschirm, Assembler und Loader als Zielsystem zu verwenden. So begann die Verbindung von Unix mit DEC-Hardware. „Space-Travel" wurde für die PDP-7 umgeschrieben, außerdem ein Assembler und ein rudimentärer System-Kernel entwickelt und auf Gecos für die PDP-7 übersetzt. Die Operation „FORK" zur Erzeugung von neuen Prozessen kam im Laufe der Zeit ebenso hinzu wie einige wesentliche Utilities, wie File Copy, Edit, Remove und Print. Das Ergebnis war ein System für zwei Benutzer. Den Namen Unix, ein Hinweis auf die vereinheitlichte Entwicklungsumgebung für mehrere Programmierer, die das System bieten sollte, erfand Brian Kernighan 1970.

Die Abteilung erwarb schließlich eine PDP-11 für ein Projekt in der Textverarbeitung. Ken Thompson hatte bis dahin das Management davon überzeugt, daß Unix ein brauchbares Betriebssystem innerhalb von Bell-Labs sein könnte; so wurde sofort mit der Portierung auf die PDP-11 begonnen. Diese Version wurde „First Edition" genannt und in einem Manual durch Thompson und Ritchie im November 1971 dokumentiert. Alle wichtigen Funktionen der modernen Unix-Systeme fanden sich in dieser Version, mit Ausnahme der „Pipes", aber mit dem Filesystem, der Prozeßverwaltung, der Systemschnittstelle („Shell", von Stephen Bourne geschrieben) und den wichtigsten Kommandos. Die Bedürfnisse der ersten Benutzer, den Mitgliedern des Dokumentations-Projekts, gaben Unix eine eindeutige Ausrichtung hin zur Textverarbeitung.

Auf Drängen von **Doug McIlroy**, eines Mitarbeiters, wurde die Shell für die Verarbeitung von Pipes umgeschrieben. Diese „Second Edition" erschien im Juni 1972. Zu dieser Zeit

ment-Library, einer Grafik-Bibliothek (GKS) und allen anderen Tools, die Unix bietet.
- Als grafische Workstation ist der InterPro-32 mit einer VAX oder MicroVAX verbunden, auf der Intergraph-Anwendungs-Software abläuft.
- Im Emulations-Modus arbeitet der InterPro-32 als DEC-VT100- oder VT220-, als Tektronix-4105- oder – über ein SNA-Gateway – als IBM-Serie-327x-Ter-
- Unter PC-DOS schließlich läuft der Computer als IBM-PC- und PC/XT-kompatibler Personal-Computer.

Unter Unix bietet Intergraph bisher nur ein CAE-Paket für den hierarchischen Schaltplanentwurf an, das als Ausgangssystem für die Leiterplatten-, IC- und Hybrid-Entwicklungs-Software unter VMS dienen kann. Man plant jedoch, in absehbarer Zeit alle für die VAX entwikkelten Pakete auch auf dem InterPro-32 unter Unix zu portieren. Daneben gibt es eine Reihe von Software-Firmen, die Applikationsprogramme anbieten.

Grafik-Workstation MG-1

Der Arbeitsplatzrechner MG-1 von Whitechapel ist mit grafischem 1000×800-Pixel-Schwarzweiß-Bildschirm ausgestattet, läuft unter Genix-4.1 mit virtueller Speicherverwaltung nach dem Demand-Paging-Verfahren und ist auf dem NS32000-Chipsatz aufgebaut. Er bietet die Leistung einer Workstation zum Preis eines anspruchsvollen Personal-Computers. Volle grafische Unterstützung, VT100-Terminal-Emulation sowie ein Ethernet-Controller sind integriert und bis zu 4 MByte Hauptspeicher vorgesehen. Über eine IBM-PC-Busschnittstelle lassen sich Standard-IBM-PC-Erweiterungskarten anschließen. Ein 0,8-MByte-Floppy-Laufwerk und eine 45-MB-Winchesterplatte gehören zur Ausrüstung dieses Rechners. Whitechapel bietet selbst keine Anwender-Software im CAD/CAM-Bereich an, offeriert jedoch einen „Third-Party-Software"-Katalog.

Datenbanken

Auf Unix-Basis gibt es zur Zeit eine große Anzahl von DBM-Systemen, die ausnahmslos nach dem Relationen-Modell aufgebaut sind. Neben so bekannten Namen wie „Ingres", „Oracle", „Informix" und „Unify" hat sich auf dem deutschen Markt mit „db++" ein Newcomer eta-

waren Kernel und Utilities immer noch in Assembler verfaßt. Parallel mit der Verbesserung von Unix arbeitete Ken Thompson an der Sprache B, einem Abkömmling der Sprache BCPL. Tatsächlich wurde der Assembler für die „Second Edition" in B geschrieben. Anfängliche Versuche, Unix in B und die verbesserte Version „NB" umzuschreiben, scheiterten. NB wurde verbessert und „C" genannt. 1973 wurde Unix erfolgreich in C umgeschrieben, wobei Thompson die Prozeßverwaltung und Ritchie das I/O-System schrieben.
Innerhalb von Bell-Labs fand Unix mehr und mehr Verbreitung, wobei auf vielen Maschinen verschiedene Versionen liefen. Im Mai 1975 wurde „Edition Six", auch bekannt als „Version 6", als erste Version offiziell außerhalb der Firma vertrieben. Das Band war PDP-11-kompatibel und für einen symbolischen Betrag ohne Garantie erhältlich; Universitäten waren demzufolge die Hauptabnehmer. Die Arbeit innerhalb von Bell-Labs ging jedoch weiter: das Filesystem wurde umgeschrieben, um größere Files speichern zu können, und die Shell zur besseren Unterstützung von Shell-Programmen modernisiert.
Das letzte größere Projekt von Thompson und Ritchie bestand darin, das Unix-System portabel zu machen. Die Zielmaschine war eine Interdata-8/32 (jetzt Perkin-Elmer), ein 32-Bit-Computer, der nach Meinung der beiden unterschiedlich genug von der PDP-11 war, um Maschinen-Abhängigkeiten aufzudecken. Das Projekt hatte auch einige Erweiterungen von C zur Folge, wie z. B. „Unions", Typendefinitionen und „Casts". Das Resultat war 1979 die „Edition Seven" oder „Version 7".
Die Universität von Kalifornien in Berkeley war eine der Ausbildungsstätten, welche die Version 7 erhielt. Die Weiterentwicklung von Unix an dieser Universität führte zu zwei verschiedenen Unix-Zweigen: die Standard-Ausgaben von Bell und die Berkeley-BSD-Versionen (Berkeley-Software-Distribution). Obwohl von Bell eine Unix-Version (Unix/32V) für die VAX-Serie von DEC erhältlich war, konnte diese die Vorteile des anforderungsgesteuerten, seitenorientierten, virtuellen Speicherkonzeptes der VAX-Architektur nicht ausnutzen. Das Institut für Computer-Wissenschaften an der Universität Berkeley modifizierte den Kernel, um so einen anforderungsgesteuerten, seitenorientierten, virtuellen Speicher auf der VAX zu implementieren. Berkeley-3.1-BSD wurde im Dezember 1979 freigegeben und an alle Besitzer einer Unix-Source-Lizenz gegen eine symbolische Gebühr ausgeliefert. Berkeley-4.1-BSD kam im Oktober 1980 heraus. Es enthält die C-Shell, die unter anderem Möglichkeiten zur Job-Steuerung und Verwaltung von begrenzten Hardware-Ressourcen bietet. Daneben ist als Verbesserung gegenüber Version 7 vor allem der Full-Screen-Editor „VI" zu erwähnen, der sich mittlerweile in der Unix-Gemeinde großer Beliebtheit erfreut. Berkeley-4.2-BSD, bei dem etwa 80 Prozent des Kernels neu geschrieben wurde, enthält gemeinsam benutzte Segmente, ein schnelleres Filesystem und eine Schnittstelle für lokale Netzwerke. Berkeley-4.3-BSD erschien Ende 1985 und enthält neben dem Xerox-Network-Systems-(XNS-)Kommunikations-Protokoll auch ein Paket für die Unterstützung von CAD/CAE.
Western Electric brachte 1982 Unix-System III heraus, wobei der Name dem Wunsch entsprang, die Unix-Versionen von Bell neu zu organisieren und neu zu numerieren. System III enthält wesentliche Verbesserungen im Kernel, neue Utilities (vor allem das Source-Code-Control-System, SCCS) und Fehlerberichtigungen. Diese Version machte das Bemühen der Firma Western Electric deutlich, Unix im kommerziellen Computerbereich zu vermarkten.
Nach System III soll Bell angeblich ein System IV besessen haben, das jedoch nie veröffentlicht wurde. Die nächste offizielle Version war schließlich System V, das im Jahr 1983 herauskam und seit Anfang 1984 in großem Stil vermarktet wird.

bliert. Das Hauptkriterium, das diese Datenbank von seinen Konkurrenten unterscheidet, ist die vollständige Integration in die Unix-Umgebung. Die C-ähnliche Syntax der Abfragesprache und die Implementierung der Datenbank als Sammlung von kleinen Tools bewirken, daß sich geübte Unix-Benutzer sofort heimisch fühlen. Die db++-Programme können ohne weiteres Bestandteil von Pipes sein und somit Ausgaben von anderen Programmen verarbeiten bzw. eigene Ausgaben an andere Programme weiterreichen. So ist es z. B. leicht, das Ergebnis von Abfragen mit der Unix-Utility „TAR" auf ein Magnetband zu kopieren.

Man geht sogar so weit, für die Installation von Zugriffsberechtigungen oder das Löschen von Relationen keine besonderen DB-Funktionen vorzusehen, sondern überläßt dies völlig den jeweiligen Unix-Kommandos (chmod und rm). Darüber hinaus ist durch ein Escape zur Shell von jedem Teil des db++-Systems aus sofort der Unix-Hintergrund verfügbar. Daß eine Datenbank nicht nur mit den „üblichen" kommerziellen Anwendungen in Zusammenhang gebracht werden muß, wollen die Hersteller (Concept Asa) mit einer Portierung auf die Grafik-Workstation MG-1 von Whitechapel beweisen. Sie sehen im Einbringen ihrer Datenbank in Grafik-Anwendungen eine ideale Kombination, die erst durch die Offenheit von db++ gegenüber Unix ermöglicht wird.

Für Anwendungs-Software unter Unix gibt es eine Reihe von Katalogen:

- Unix-Produkt-Katalog der /usr/group-Vereinigung,
- Unix-Micros-Katalog der EUUG (European Unix Users Group),
- Unix-System-V-Software-Katalog (AT&T),
- Software-Börse Sinix (Siemens),
- ISIS-Software-Report,
- Ultrix-Software-Guidebook (Digital Equipment),
- ISV-Kataloge der Halbleiter-Hersteller.

Echtzeit-Anwendungen

Ein Betriebssystem, das Echtzeit-Anwendungen unterstützen soll, muß zumindest folgende Eigenschaften aufweisen:
- Schnelle Antwortzeit auf Hardware-Interrupts. Das impliziert, daß Prozesse oder zumindest Teile davon im Speicher fixiert werden können.
- Schnelle Erfassung von großen Datenmengen, was ein Filesystem mit großen zusammenhängenden Files erfordert, um eine kontinuierliche Abspeicherung der Werte gewährleisten zu können.
- „Context-Switching" und „Scheduling" müssen den Echtzeit-Bedürfnissen angepaßt sein. Das heißt im einzelnen, die Zeit für die Kontext-Umschaltung muß minimal sein und es muß im Kernel mindestens zwei Datenstrukturen für das Scheduling der Prozesse geben: eine für normale Timesharing-Prozesse und eine für Echtzeit-Prozesse.

Die erste dieser Forderungen kann nur von einigen Unix-Versionen erfüllt werden. Das Unix-Filesystem benutzt für die Abspeicherung von großen Files „links", also Zeiger zu anderen Plattenblöcken. Dies macht eine konsistente Übertragungsrate unmöglich, da die genaue Anzahl von Plattenzugriffen nicht vorhersehbar ist. Das Context-Switching von Unix ist nicht für Echtzeit-Anwendungen gebaut.

Hält man sich vor Augen, zu welchem Zweck Unix ursprünglich geschrieben wurde, nämlich hauptsächlich für Text-Vorbereitung und -Verarbeitung, so kann es nicht verwundern, daß das Betriebssystem für Echtzeit-Anwendungen nicht geeignet ist. Ohne tiefgreifende Änderungen im Kernel und in der Unix-System-Philosophie ließe sich eine solche Möglichkeit nicht vorsehen. Zwar gab es von AT&T in der Vergangenheit schon Verlautbarungen, daß eine der nächsten Releases auch Echtzeit-Merkmale enthalten werde, bisher blieb es jedoch bei den Ankündigungen. Darüber hinaus war hiermit wohl „Transaction-Processing", eine „abgemilderte" Form von Echtzeit-Verhalten, gemeint. Wer bei Echtzeit-Anwendungen nicht auf Unix verzichten will, dem bieten sich zwei Alternativen:
- Auslagerung der Echtzeit-Aufgaben auf ein intelligentes Subsystem. Dieses ist mit ausreichendem Pufferspeicher ausgerüstet und sendet die Daten via Netzwerk oder Bus-Verbindung dem unter Unix laufenden Hostrechner. Diese Art der Aufgabenverteilung wird auch bei Applikationen notwendig, die normalerweise nicht in die Kategorie „Echtzeit" fallen. So ist es bei interaktiven Grafik-Anwendungen notwendig und üblich, von der Maus gelieferte Interrupts von einem Coprozessor abarbeiten zu lassen. Die möglicherweise lange Reaktionszeit des Unix-Systems verlangsamt die Bildschirmdarstellung.
- Die Entscheidung fällt deshalb in der Regel für eines der zahlreichen Unix-ähnlichen Systeme mit Echtzeit-Möglichkeiten. Dabei gibt es Systeme, die durch mehr oder weniger tiefgreifende Modifikationen des Unix-Kernels entstanden sind, wie etwa RTU von Masscomp und solche, die von Grund auf neu geschrieben wurden, jedoch eine Unix-Oberfläche aufweisen. Ein Vertreter dieser Gruppe ist „Unos" von Charles River Data Systems. Einige der Systeme weisen Kompatibilität zu System V auf, wie z. B. „Regulus" von Alcyon, das auch von AT&T lizenziert wurde. Auch Unos soll kompatibel mit den System-V-Systemaufrufen sein, während man bei Quantum Software Systems auf die Einhaltung des /usr/group-Standards bei dem realzeitfähigen QNX verweist. AT&T selbst hat mit „MERT" (Multi-Environment-Real-Time-System) einen System-Emulator, der dem emulierten System Echtzeit-Eigenschaften verschafft.

Unix für kommerzielle Anwendungen

System V oder 4.2 BSD?

System V ist die modernste, umfangreichste und beste Version des Betriebssystems Unix auf dem Markt und in der Lage, alles auf der Welt besser, schneller und sauberer zu erledigen. Das behauptet AT&T. Unix 4.2 BSD ist das anspruchsvollste, flexibelste, eleganteste und gradlinigste Betriebssystem, was es überhaupt gibt. Diese Aussage hört man von 4.2-Anwendern. Können beide Versionen die beste sein?

Hier findet ganz offensichtlich eine Art Familienfehde statt. Der Stammbaum von Unix-Systemen zeigt einige Merkmale, die sich auch in der griechischen Mythologie finden, nämlich Inzest und Ödipuskomplex. Die Historie läßt sich kurz zusammenfassen: Seit der „Urzeit" von Unix, als Ken Thompson ein besseres Computerspiel suchte, entwickelten sich zwei parallele Linien: Eine in den Bell Labs, die andere in verschiedenen Universitäten, die Unix gegen relativ geringe Lizenzgebühren erhielten und zum Gegenstand wissenschaftlicher Untersuchungen machten. Obwohl viele Hochschulen mit Unix experimentierten, setzte nur die University of California in Berkeley Standards, indem sie die berühmten BSD-Versionen von Unix herausgab; Modifikationen des Betriebssystems, die die Computer Science Research Group erarbeitete.
Die Komplexität des Themas „Unix" wird offensichtlich, wenn man sieht, an welchen Punkten sich die beiden Familienzweige gegenseitig befruchtet haben. Ein Vergleich zwischen den Stammbäumen von Fürstenhäusern zwingt sich hier direkt auf: Beziehungen untereinander sind zwar nicht direkt offensichtlich, aber von größter Bedeutung. Innerhalb von Bell Labs teilte sich die Version 6, die es seit 1975 gibt, in drei Richtungen auf:

- Ein internes Forschungs- und Entwicklungssystem der USG bei Bell (Unix Suport-Group)
- Version 7, die 1979 herauskam (Wenn Sie eine PDP-11 haben, auf der Unix läuft, dann handelt es sich um diese Version),
- Unix 32 V (Eine „leicht virtuelle" Version der Version 7 für die VAX, die als Basis für Berkeley 3.0 BSD diente).

Anfang 1982 kam von AT&T die erst kommerzielle Implementierung von Unix, das System III. Dieses war mit sehr hübsch gedruckten Handbüchern ausgestattet und Verbesserungen, die ein kommerzieller Benutzer sich wünscht, außerdem mit Kommandos, die die Systemverwaltung erleichtern. Die Einführung von „Named Pipes", die eine rudimentäre Einrichtung zur Kommunikation zwischen Prozessen darstellen, und das neue „initab", das zu Änderungen von Zuständen dient und die Existenz von anderen Zuständen als Single- und Multi-User-Mode voraussetzt, stellen die wichtigsten technischen Verbesserungen von System III dar.
Anfang 1983 kam von AT&T das System V (System IV, falls es jemals existierte, muß im „Bit-Abfalleimer" gelandet sein). Anfang 1984 gab es davon die Version „Release 2.0". Bei Release 2.0 setzt sich der Trend fort, Unix für kommerzielle Benutzer besser anwendbar zu machen.

Abkürzungen

AT&T: Die Abkürzungen Bell Labs, Western Electric, AT&T Technologies und alle die Firmen, die früher unter der Bezeichnung „Bell" zusammengefaßt waren
DARPA: Defence Advanced Research Projekt and Administration. Diese Leute haben das Geld
CSRG: Computer Science Research Group. Diese Leute bekommen das Geld und leisteten die Arbeit für die Entstehung von Berkeley BSD
BSD: Berkeley Software Distribution
USG: Unix Support Group bei Bell Labs

Wenn man also zurückblickt, ergeben sich folgende Fakten: Bis AT&T sich dieser Sache annahm, war Unix ein Entwicklungswerkzeug für den Forschungs- und Entwicklungsbereich, das aufgrund seiner Natur geradezu zum „Herumbasteln" aufforderte. Als Resultat dieser Bastelei ergab sich ein sehr arbeitsfähiges Betriebssystem: Es bietet eine Umgebung, in der jeder, auch ein kommerzieller Anwender, irgend etwas anfangen kann. Auf der anderen Seite kann ein kommerzieller Anwender keine Ausfallzeiten oder einen Systemzusammenbruch tolerieren, nur weil ein Computergenie einen Geistesblitz aufgrund des Konsums einer Anchovispizza hatte. Dann empfiehlt es sich, Release 2.0 zu wählen, denn diese Version bietet neue Dienstprogramme sowie eine „glatte Dokumentation", Standardsinform, standardisierter Disk- und Tape-Name sowie das Versprechen von AT&T, daß zukünftige Modifikationen am Unix-Kern, Programme, die auf dem derzeit verfügbaren Kern laufen, nicht betreffen.

Der Unterschied zwischen 4.1 BSD und 4.2 BSD

In der Zeit, als AT&T gewisse Veränderungen durchzustehen hatten, machte Unix in Berkeley eine Metamorphose anderer Natur durch. Während die frühen Unix-Versionen für eine lange Periode der Entwicklung auf Rechnern vom Typ PDP-11 liefen und obwohl AT&T eine „virtuelle" Version Unix für die VAX hervorbrachte, kam aus dieser Richtung bisher keine Ausführung für den virtuellen Speicherzugriff nach dem Demand-Paged-Verfahren für die VAX. Die Computer Science Research Group (CSRG), die von der Defence Advanced Research Projekt Administration (DARPA) gegründet wurde, brachte in diese Entwicklung die Energie, den Geist und die freie Art des Denkens mit ein, die im akademischen Bereich im Überfluß vorhanden ist. Das Resultat war Ende 1979 Berkeley 3.0 BSD, darauf folgte 1980 die Version 4.0, die wiederum noch im gleichen Jahr zur Ausführung 4.1 modifiziert wurde. Diese letzte Version stellt ein sehr ausgefeiltes stabiles Betriebssystem dar, das im Gegensatz zu den AT&T-Versionen die C-Shell mit Job-Control bieten, außerdem den Full-Screen-Editor „vi" („Standard"-Unix bietet bis System V lediglich einen zeilenorientierten Editor), Pascal, Lisp und etwa 80 neue Kommandos. Während die AT&T-Versionen für die VAX immer noch mit den Hardware-Restriktionen der PDP-Rechner belastet sind (z. B. die 1-K-Blockierung), brachte Berkeley Unix in das technische Zeitalter der VAX. Man muß wirklich einmal sagen, daß Berkeley sehr gewissenhaft daran gearbeitet hat, 4.1 als reifes, solides Betriebssystem herauszubringen, während gleichzeitig die wissenschaftlichen Untersuchungen mit sehr großen Anstrengungen weiterverfolgt wurden.

Die Version 4.2 erblickt das Licht der Welt

„Die Idee bei 4.2 BSD war, sich von irgendeinem System zu lösen und zu überlegen, was ein Anwender wirklich benötigt. Das Ziel war, eine neue Basis zu bieten – alles was neu definiert werden mußte, auch wirklich neu zu defi-

nieren. Dann gibt es anstelle einer konstanten Weiterentwicklung, Veränderung und Anpassung einen einzigen Umbruch."
So die Meinung von Mike Karels, „Oberprogrammierer" der CSRG.
Tatsächlich handelt es sich beim 4.2 um ein vollständig überarbeitetes Betriebssystem. CSRG ging vom Code aus, der unverändert der Version 6 entstand und entwickelte diesen im Licht der Anforderungen größerer und komplexerer Computer mit vielen Benutzern. Die Version 4.2 unterscheidet sich sehr stark von vorhergehenden Berkley-Versionen, insbesondere wegen der Fähigkeit des Kerns für die LAN-Kommunikation und dem schnellen Dateisystem, das den Disk-Blocking-Faktor auf 4 K erhöht. In gleicher Weise, wie 3.0 die erste Version in einer Serie von Releases darstellte, die wesentliche neue in den folgenden Versionen verbesserte, Konzepte einführte, begann mit 4.2 die Einführung neuerer, verbesserter Konzepte, die in zukünftigen Ausführungen weiter weiterentwickelt werden. CSRG arbeitet derzeit an Verbesserungen des Kerns sowie an neuen Anwendungen, z. B. „Windowing" und virtuelle Dateisysteme.

System V

Unser Anliegen ist es, Unix als Softwareumgebung für ein breites Spektrum von Mikro- und Großcomputern durchzusetzen. Diese Philosophie hat einige sehr wichtige Konsequenzen: Einserseits müssen die Investitionen des Kunden für existierende Unix-Anwendungsprogramme geschützt werden, andererseits muß die Aufwärtskompatibilität in möglichst hohem Maße gewährleistet bleiben. „Eine offene Architektur und universeller Quellencode des Unix trägt dazu bei, daß dieses Betriebssystem für eine möglichst große Vielfalt an Hardware brauchbar ist." So Doug Kevorkian, Bell Technical Labs.
Jetzt haben wir also System V, Release 2.0 und BSD 4.2. Obwohl es eine Menge brillanter und genialer Denker bei AT&T gibt, die exzellente Arbeit auf dem Gebiet der Computerwissenschaft leisten, liegt der Schwerpunkt des Interesses dieser Firma nun einmal darin, mit Unix Dollars zu machen. Die Versionen von AT&T tendieren daher in Richtung solider Merkmale, die für kommerzielle Anwendungen brauchbar sind. Das Hauptanliegen von AT&T ist Geld zu verdienen. Und auch diejenigen, die kaufen, wollen Geld verdienen. Und sie wollen ihr Geld für ein Produkt ausgeben, von dem sie meinen, daß es komfortabel ist.
Wenn man sich System V und Release 2.0 einmal genau anschaut, erkennt man die Einflüsse von Berlekey sowie die Verbesserungen nach Berkeley-Art: z. B. der Editor „vi", eine neue Form der Shell, die die Jobsteuerung übernimmt, ein Paket mit der Bezeichnung „Curses", das auf „Terminfo" der Purdue University basiert, sowie Shell-Funktionen, die verallgemeinerte Formen von „Aliases" der C-Shell sind. In ähnlicher Weise entsprach AT&T den Anforderungen des Marktes bei der virtuellen Speicherverwaltung nach dem Demond-Paged-Verfahren.
AT&T zeigt damit, daß man bei dieser Firma sehr sensibel auf den Bedarf nach ausgezeichneten Produkten, die im Forschungsbereich entwickelt wurden, reagiert. Das technische Management bei Bell zeigt großen Respekt vor den Kollegen im akademischen Bereich. Allerdings ist trotz vieler Verbesserungen die derzeitige Version von AT&T für manche Anwendungen nicht so gut geeignet. So ist z. B. die Netzwerkfähigkeit von System V derzeit eingeschränkt. Während die Version 4.2 der Netzwerkkommunikation weitgehend entgegenkommt, in dem es TCP/IP und andere Protokolle unterstützt, haben die Entwickler von System V momentan nur die kurzfristigen Bedürfnisse ihrer Kunden befriedigt.
„3BNet" das im Juli 1984 auf der NCC als Ethernet-kompatibel vorgestellt wurde, verbindet IBM/PC-Benutzer, die gemeinsam auf Dateien zugreifen wollen oder mit einem zentralen AT & T-Computer zusammenarbeiten müssen. Das 3B-Softwareinterface für die PCs befindet sich direkt auf dem System V und ermöglicht es einem Benutzer, zwischen Terminalemulation auf der zentralen Unix-Maschine und dem ursprünglichen PC-Betriebssystem (DOS) umzuschalten. Eine oder mehrere der DOS-Directories können physikalisch in der zentralen Unix-Maschine residieren, und ein Programm, das in der DOS-Umgebung läuft, kann Dateien benutzen, die auf der Unix-Maschine sind. Das ist bestimmt sehr interessant, wenn man mit PC/DOS arbeitet, kann aber keinesfalls mit einer 4.2-Maschine in einer Netzwerkkonfiguration, die Dateien mit Lichtgeschwindigkeit transferiert, verglichen werden.
Welches Fazit kann man nun daraus ziehen? AT&T hat sich klugerweise die „Früchte" einverleibt, die im Garten der Universitäten gewachsen sind. Aufgrund der einzigartigen Natur eines portablen Unix hat der Lizenzgeber für den Quellencode allerdings einen etwas ungewöhnlichen Standpunkt. Die Systemhersteller, die sich an AT&T halten und deren Unix-„Standard" übernehmen, müssen AT&T vertrauen, daß diese Firma die richtigen Entscheidungen bei der Auswahl von Modifikationen aus dem Universitätsbereich trifft, daß die AT&T-Forschungsabteilung aktiv bleibt und daß die Früchte dieser Anstrengungen auch dem Benutzer zu gute kommen. Man muß auch darauf vertrauen können, daß AT&T Verbesserungen an Unix unter dem allgemeinen Gesichtspunkt und nicht nur für die eigenen Computer von AT&T vornimmt. Der Systemhersteller kann natürlich auch Berkeley BSD als Ausgangspunkt für eine Software-Produktlinie akzeptieren und Innovationen von anderen Quellen erwerben. Beispiele dafür sind DEC, Sun oder National Semiconductor.
Es läßt sich vielleicht am besten so verdeutlichen: Unix ist nicht nur deshalb interessant, weil es elegante Konzepte verwirklicht, sondern Unix ist interessant, weil es portabel und in Form von Quellencode verfügbar ist. Deshalb ist es auch modifizierbar: Man kann alles daraus machen, was man will. Wenn es aber zu einem „Standard" wird. Ist es dann noch Unix?

Kathleen B. Wallache

Berkeley-Unix für Supermikro

National Semiconductor hat die Unix-Version 4.2 bsd auf seine Mikroprozessor-Familie 32000 portiert. Das unter der Bezeichnung Genix 4.2 vertriebene Betriebssystem unterstützt auch den Anschluß an lokale Netzwerke. Nach Auskunft des Anbieters soll diese Betriebssoftware vor allem in Workstations der oberen Leistungsklasse eingesetzt werden. National bietet nun alle wichtigen Unix-Versionen für seine Supermikros an, einschließlich System V von AT&T.
Die Unix-Version 4.2 ist keine Weiterentwicklung der Version 4.1, sondern wurde an der Berkeley-Universität neu konzipiert. Die Netzwerkfähigkeit der Version 4.2 basiert auf dem TCP/IP-Protokoll, das im Auftrag des US-Verteidigungsministeriums entworfen wurde. Außerdem enthält das Betriebssystem eine Weiterentwicklung des „Fast File System", welches einen schnelleren Zugriff auf die Plattenspeicher erlaubt. Die virtuelle Speicherverwaltung (demand paging) wurde speziell an den 32-Bit-Mikroprozessor angepaßt.

Stan Baker

Günther Hausmann

Struktur und Funktionen des Echtzeit-Betriebssystems VRTX/Series-32000

Obwohl sich durch das Erscheinen der modernen 32-Bit-Mikroprozessoren die Aufmerksamkeit auf Unix konzentriert hat, gibt es eine Vielzahl potentieller Applikationen, wo Unix nicht die einzige oder beste Wahl eines Betriebssystems darstellt. Besonders Applikationen in den Bereichen Robotertechnik und Prozeßsteuerung erfordern Betriebssysteme, die speziell dafür entwickelt sind, Echtzeit-Funktionen auszuführen, die charakteristisch für solche Anwendungen sind.

Kernel bringt Vereinfachung

Unabhängig von der Verschiedenheit der Applikationen weisen alle Echtzeit-Systeme gemeinsame Merkmale auf und alle Echtzeit-Betriebssysteme setzen daher Grundfunktionen wie „Task-Scheduling" und „Intertask-Kommunikation" voraus. Im besonderen ist es erforderlich, daß solche Systeme eine Vielzahl von Tasks parallel verarbeiten und per Interrupt gesteuert werden können. Die Entwicklung von Echtzeit-Betriebssystemen läßt sich durch den Einsatz eines „Kernel" (Echtzeit-Ablaufsteuerung), der die Parallel-Verarbeitung steuert und andere Grundfunktionen bereitstellt, stark vereinfachen.

In den vergangenen Jahren ist ein solcher Kernel, namens VRTX der Firma Hunter and Ready Inc., zum De-facto-Industrie-Standard geworden, und verschiedene Versionen wurden auf eine Reihe von 8-Bit- und 16-Bit-Mikroprozessoren implementiert. VRTX/Series-32000 ist die erste Implementation für 32-Bit-Mikroprozessoren. Mit ihren E/A-, File-Management- und Debugging-Software-Teilen stellt diese Implementation dem Entwickler bewährte Software (VRTX, IOX, FMX, TRACER) zur Verfügung, der die Hardware-Bausteine der 32000-Familie ergänzt.

Variable Software-Nutzung

Die Rolle des VRTX/Series-32000 ist es, ein Interface zwischen der Hochsprachen-Applikations-Software und der Hardware zu bieten, das es dem Applikationspro-

Tabelle 1

SC_TCREATE	Task Create: creates a new task assigning it a priority and an ID.
SC_TDELETE	Task Delete: deletes a task.
SC_TSUSPEND	Task Suspend: suspends execution of a task.
SC_TRESUME	Task Resume: resumes execution of a suspended task.
SC_TPRIORITY	Task Priority Change: changes the priority of a task.
SC_TINQUIRY	Task Inquiry: returns the status and priority of a task.
SC_LOCK	Disable Task Rescheduling: makes the running task unpreemptible.
SC_UNLOCK	Enable Task Rescheduling: makes the running task preemptible again.
SC_TSLICE	Enable Time-Sliced Scheduling: turns time-slicing on (or off).

grammierer erlaubt, die Funktionen zu nutzen, ohne sich um deren Implementierung auf Maschinenebene kümmern zu müssen. Dieses Echtzeit-Betriebssystem wurde speziell als Software-Modul entwickelt, das auf einfache Weise in eine Vielzahl von Systemen ohne Modifikationen integriert werden kann. Wie jedes andere Modul muß man das VRTX/Series-32000 in bezug auf seine Funktionsmöglichkeiten, Verarbeitungsgeschwindigkeit und Zuverlässigkeit beurteilen. Hinsichtlich der Funktionalität ist das VRTX/Series-32000 identisch mit den VRTX-Versionen, die bislang auf einfacheren Mikroprozessoren implementiert wurden. Die Vorteile der 32000-Architektur führen jedoch im Vergleich dazu zu einer wesentlichen Steigerung der Leistungsfähigkeit. Die Software-Zuverlässigkeit wird dadurch garantiert, daß derselbe VRTX-Quellcode portiert wurde, der schon in einer Vielzahl von Applikationen erprobt worden ist.

Die wichtigste Fähigkeit des VRTX/Series-32000 ist die Parallel-Verarbeitung in Multitasking-Applikationen. Wie die Prozedur in konventionellen Programmen, ist die Task die Basiseinheit der Echtzeit-Software. Tasks können mit jeder anderen Task kommunizieren, sie sind jedoch auf jeden Fall logisch getrennt voneinander und können parallel abgearbeitet werden. Das VRTX/Series-32000 bietet die drei Schlüssel-Software-Elemente eines Multitasking-Systems:
– Task-Scheduling,
– Kommunikation zwischen den Tasks,
– dynamische Speicherzuweisung.
Darüber hinaus unterstützt das System echtzeitgetaktete E/A- und Interrupt-Handling-Funktionen.

Die Funktionen des VRTX/Series-32000 werden mit Hilfe von System-Calls aufgerufen, die als High-Level-Erweiterungen des Befehlssatzes der 32000-Serie arbeiten. *Tabelle 1* zeigt als Beispiel eine Aufstellung der Task-Management-System-Calls. Systemaufrufe können mit Hilfe einer Hochsprache unter Verwendung von Bibliotheksroutinen, die eine korrekte Parameter-Übergabe sicherstellen, erstellt werden, oder sie werden in Assemblersprache unter Einsatz eines Trap-Befehls geschrieben.

Task-Management

Obwohl die Tasks definitionsgemäß parallel abgearbeitet werden sollen, kann ein Einzel-Prozessorsystem natürlich nur eine Task zeitgleich verarbeiten, so daß die CPU mit Zeitmultiplex-Verfahren zwischen den parallel zu verarbeitenden Tasks hin- und hergeschaltet werden muß. Dieses Verfahren, als „Task-Scheduling“ bekannt, läuft im Hintergrund des Systems ab und ist nicht von der Applikation abhängig. Der Anwenderprogrammierer sollte sich deshalb nicht um diese systeminterne Funktion kümmern müssen. Der „Task-Scheduler“, der Bestandteil des VRTX/Series-32000-Systems ist, wickelt automatisch das Umschalten auf die einzelnen Tasks in einer Weise ab, die es dem Anwendungsprogrammierer erlaubt, das Umschalten zu steuern, ohne sich um die Durchführung kümmern zu müssen.

Das VRTX/Series-32000-System arbeitet mit einer frei wählbaren, prioritätsorientierten Scheduling-Technik, die flexibel genug ist, um praktisch allen Applikationen gerecht zu werden. Zum Zeitpunkt ihrer Erstellung wird jeder Task eine Priorität zugewiesen. Falls mehr als eine Task zur Verarbeitung bereitstehen, so wird diejenige mit der höchsten Priorität hierfür ausgewählt. Mehr noch: Falls das Ergebnis eines Systemaufrufs eine Task mit höherer Priorität zur Verarbeitung ansteht, so zieht das VRTX/Series-32000-System automatisch die Task mit der höchsten Priorität vor, indem es den Ablaufplan ändert und den Status der Task mit der geringeren Priorität von „Executing“ auf „Ready“ ändert.

Der Vorteil dieser Technik besteht darin, daß die Antwortzeit auf eine Echtzeit-Anforderung minimiert wird. Eine Task, die als „dringend“ definiert ist, kann sofort bearbeitet werden, wenn sie im „Ready“-Status ist. Eine einfache prioritätsorientierte Steuerung hat jedoch den Nachteil, daß rechenintensive Tasks mit hoher Priorität solche mit niedrigerer Priorität für lange Zeit abblocken können. In den meisten Systemen gibt es eine obere Grenze für die Zeit, die eine Task mit niedriger Priorität auf die „Wartebank“ geschoben werden kann.

Das VRTX/Series-34000-System löst dieses Problem auf zwei Ebenen, wobei auf keiner der beiden Ebenen der Anwendungsprogrammierer erforderlich ist, um den „Scheduler“ zu modifizieren. Als erstes kann einer Reihe von Tasks eine gleiche Priorität zugewiesen werden, so daß die CPU-Zeit auch in gleicher Weise unter diesen Tasks aufgeteilt wird. Tasks, die somit eine CPU-Zeit zugewiesen bekommen haben, können weiterhin von einer Task mit höherer Priorität ausgeschaltet werden, und der gesamte CPU-Zeit-Verteilungsprozeß läßt sich durch einen Systemaufruf ein- und ausschalten.

Dieses Verfahren bietet eine ausreichende Flexibilität, um die meisten statischen Prioritätszuweisungen, von Systemen, bei denen alle Tasks gleiche Priorität haben, bis zu solchen, wo jede Task eine unterschiedliche Priorität hat, zu meistern. Darüber hinaus gibt es jedoch noch eine zweite Ebene der Steuerung über die dynamische Prioritätszuweisung. Wie Tabelle 1 zeigt, ermöglicht der Systemaufruf „SC TPRIORITY“ den jederzeitigen Wechsel der Prioritätsebene einer Task. Falls durch das Ergebnis eines solchen Wechsels eine Task die höchste Priorität erhält und sie zur Verarbeitung ansteht, so wird sie unmittelbar ablaufen und dabei die ursprünglich aufgerufene Task in den Wartestatus versetzen.

Die dynamische Prioritätszuweisung ist eine erheblich komplexere Technik als die relativ einfache statische Priorisierung, und die Bedingungen, unter denen die Prioritäten sich ändern, variieren gemäß der jeweiligen Applikation. Aus diesem Grunde ist die Implementation eines dynamischen Prioritäten-Algorithmus in vollem Maße Aufgabe des Anwendungsprogrammierers.

Das VRTX/Series-32000-System stellt jedoch in Form von „Software-Hooks" Mittel zur Verfügung, die den Anwendungsprogrammierer bei der Generierung der Informationen unterstützen, die er benötigt, um eine spezielle Strategie zu implementieren. Zum Beispiel kann der Programmierer auf diese Weise eine Routine hinzufügen, damit die Zeit gemessen werden kann, die verging, seit eine Task zum letzten Mal bearbeitet wurde, um eventuell deren Priorität zu erhöhen, falls notwendig.

Intertask-Kommunikation

Wie das „Scheduling" ist auch die Kommunikation zwischen den Tasks ein Erfordernis bei den meisten Echtzeit-Systemen und unterliegt einer allgemeinen Lösung. In dem VRTX/Series-32000-System ist die Intertask-Kommunikation mit Hilfe von „Messages", „Mailboxes" und „Queues" (Warteschlangen) implementiert. Die Vier-Byte-Message ist die Basiseinheit der Kommunikation zwischen den Tasks. Sie kann als 32-Bit-Integer-Binär-Wort, als Fließkomma-Zahl oder als Zeiger in einer mehr allgemeinen Datenstruktur gemäß der speziellen Applikation interpretiert werden.

Messages zwischen den Tasks werden mit Hilfe von Mailboxen und Warteschlangen ausgetauscht. Bei der Ablage einer Information in einer Mailbox braucht die sendende Task nicht abzuwarten, bis die empfangende Task für die Aufnahme der Nachricht bereit ist. Eine Mailbox kann zur gleichen Zeit nur eine Nachricht aufnehmen, aber in zahlreichen Applikationen gibt es Abläufe, bei denen Tasks die Messages schneller erzeugen, als sie dann empfangen werden können. In diesen Fällen lassen sich die Messages in einer Warteschlange deponieren, anstatt in den Mailboxen.

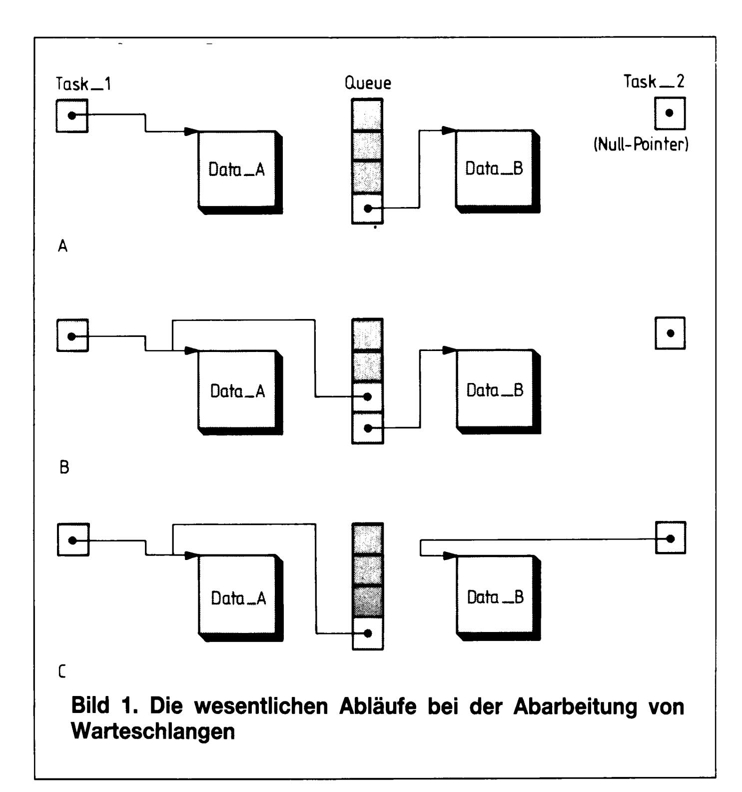


Bild 1. Die wesentlichen Abläufe bei der Abarbeitung von Warteschlangen

Tabelle 2

SC_POST	Post Message: places a message in a mailbox. If the mailbox is full, the caller is notified.
SC_PEND	Pend For Message: obtains a message from a mailbox. If the mailbox is empty, the task is suspended until a message arrives or a specified time interval passes.
SC_ACCEPT	Accept Message: obtains a message from a mailbox. If the mailbox is empty, the caller is notified.
SC_QCREATE	Create Message Queue: creates a new queue.
SC_QPOST	Post Message To Queue: adds a message to the end of a queue; notifies the caller if the queue is full.
SC_QPEND	Pend For Message From Queue: obtains the message at the front of a queue. If the queue is empty, the task is suspended until either a message arrives or a specified time interval passes.
SC_QACCEPT	Accept Message From Queue: obtains the message from the front of a queue; the task is notified if the queue is empty.
SC_QINQUIRY	Inquire As To Queue Status: returns the count of messages currently in the queue, and the status of the queue.

Eine Warteschlange, wie sie *Bild 1* zeigt, kann eine bestimmte Zahl von Messages aufnehmen. Die Länge der Warteschlange wird bei deren Erzeugung definiert. Messages werden ans Ende der Warteschleife angefügt und vom Anfang entnommen. Der Vorteil, der sich daraus ergibt, daß die Möglichkeit besteht, Messages zwischenzuspeichern, muß gegen die längere Zeit abgewogen werden, die daraus entsteht, daß man mit Warteschlan-

gen arbeitet. Das VRTX/Series-32000-System bietet beide Möglichkeiten und läßt dem Anwender die Wahl.

Tabelle 2 zeigt die Systemaufrufe für die Intertask-Kommunikation. Merke: es gibt zwei Möglichkeiten für den Empfang einer Message von einer Mailbox oder aus einer Warteschlange, indem man entweder die „PEND“- oder die „ACCEPT“-Funktionen benutzt. PEND veranlaßt die empfangende Task, auf die erwartete Message zu warten, sofern die Mailbox oder die Warteschlange leer ist. Bei der ACCEPT-Funktion wird die Task nicht ausgesetzt, aber informiert, daß keine Message vorhanden ist.

Eine Mailbox entspricht einer binären Semaphore, wie sie im allgemeinen für die Task-Synchronisation benutzt wird. Die Systemaufrufe können ebenfalls zur Synchronisation von Tasks, zusätzlich zur Übertragung von Daten zwischen ihnen, benutzt werden. Bei der Synchronisation von Abläufen ist der Inhalt der Message nicht wichtig, statt dessen führen die PND- und POST-Aufrufe dieselben Funktionen aus wie die klassischen „Wait“- und „Signal“-Semaphoren.

Speicherverwaltung

Nicht durch die Verwaltung eines virtuellen Speichers in Konfusion gebracht, hat die Speicherverwaltung des VRTX/Series-32000-Systems die Aufgabe, bestimmte Speicherbereiche den jeweiligen Tasks zuzuweisen. Obwohl es häufig möglich ist, einer Task einen festen Speicherbereich zuzuordnen, so gibt es trotzdem zahlreiche Fälle, wo die Größe des erforderlichen Speichers nicht konstant ist. Anstatt unnötig Speicherplatz zu verschwenden, in dem jeder Task das Maximum an Speicherkapazität zugewiesen wird, das erforderlich sein könnte, erlaubt das VRTX/Series-32000-System die dynamische Anpassung des Tasks-Speichers während des Programmablaufs.

Das System regelt die dynamische Speicherzuweisung durch Verwaltung eines freien Pools an Speicherblöcken. In den meisten Echtzeit-Multitasking-Applikationen ist es wichtig, daß einer Task jeder zusätzlich erforderliche Speicherraum, der erforderlich ist, in einer sehr schnellen und vorhersehbaren Weise zugeordnet wird. Dies setzt eine Struktur voraus, die auf Blöcken mit variabler Größe basiert. Strukturen, die jedoch nur eine Speicherblockgröße zulassen, führen zur Verschwendung von Speicherplatz als ein negatives Ergebnis dieser Fragmentierung. Um dieses Problem zu lösen, unterteilt das VRTX/Series-32000-System den verfügbaren Speicher in Bereiche, wobei jeder Bereich Blöcke derselben Größe enthält, wie *Bild 2* zeigt.

In *Tabelle 3* sind die Memory-Management-System-Calls aufgelistet, die alle selbsterklärend sind. Merke: die Wahl der Blockgröße obliegt vollständig dem Programmierer. Das System selbst setzt keine Begrenzungen für die Blockgröße. Ein Bereich kann durch Hinzufügung von weiteren Blöcken erweitert werden, ohne daß die Erweiterung in einem zusammenhängenden Speicherbereich stattfinden muß.

Andere Service-Funktionen

Die übrigen Standard-Service-Funktionen des VRTX/Series-32000-Systems unterstützen Echtzeit-Taktsteuerungen, einfache Zeichen-I/O-Operationen und Interrupts. Die Zeit-Grundeinheit im VRTX/Series-32000-

Tabelle 3

SC_GBLOCK	Get Memory Block: obtains a block of free memory from a partition.
SC_RBLOCK	Release Memory Block: returns a block of memory to a partition so it can be reused.
SC_PCREATE	Create Memory Partition: creates a partition.
SC_PEXTEND	Extend Memory Partition: extends a partition by adding more blocks to it.

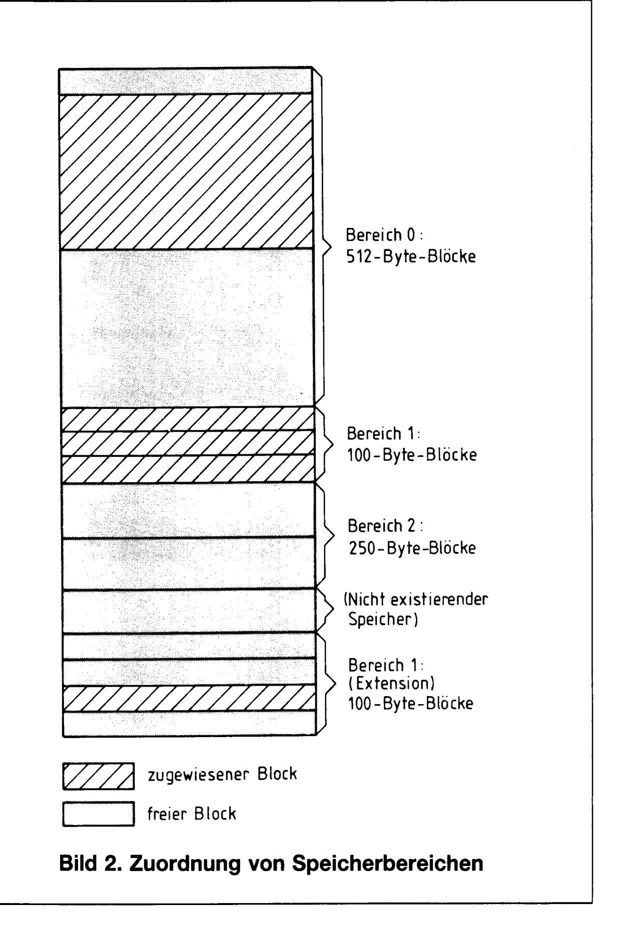


Bild 2. Zuordnung von Speicherbereichen

System wird „Tick“ genannt. Der „Tick“-Zähler wird durch den UI-TIMER-Systemaufruf, ausgelöst als Antwort auf Interrupts, die durch den Hardware-Zeitgeber erzeugt werden, inkrementiert. Der VRTX/Series-32000-Systemtakt ist deshalb ein Zeittakt mit hoher Auflösung, der die Zeit mißt, die von einem durch den Anwender gesetzten „Zeitnullpunkt“ vergeht. Es ist ein relativ einfaches Unterfangen, diesen Takt in einen absoluten Zeit-/Kalender-Wert im Rahmen der Anwender-Software umzusetzen. Wie in *Tabelle 4* zu sehen ist, erlaubt der Echtzeit-Systemaufruf ein Ablesen der Zeit oder die Initialisierung einer gesperrten Task, die für ein definiertes Zeitintervall suspendiert war.

Weil das VRTX/Series-32000-System als eine effiziente Software-Komponente ausgelegt ist, unterstützt es nur Basis-Zeichen-I/O-Operationen. Umfassende I/O-Möglichkeiten bietet das IOX/Series-32000. Weil zahlreiche Echtzeit-Systeme mit Tastaturen und Bildschirmen als Bedienungsinterface ausgestattet sind, unterstützt das VRTX/Series-32000-System diese Ein- und Ausgabegeräte direkt.

Drei Systemaufrufe sind für die Zeichen-I/O-Operationen vorbehalten: SC GETC liest das jeweils nächste Zeichen aus dem Eingabe-Einheiten-Speicher. Wenn der Zwischenspeicher leer ist, so wartet die Task, bis ein Zeichen von der Eingabe-Einheit kommt. SC WAITC veranlaßt die aufgerufene Task zu warten, bis ein definiertes Zeichen von der Eingabe-Einheit kommt. SC PUTC sendet ein Zeichen an den Ausgabe-Einheiten-Speicher. Ist dieser Speicher voll, so wird diese Task ausgesetzt, bis der Zwischenspeicher wieder in der Lage ist, Zeichen aufzunehmen.

Obwohl vorausgesetzt wird, daß ein zeitgebergesteuerter Interrupt zur Verfügung steht, um den „Tick“-Zähler auf den jeweils neuesten Stand zu bringen, so nutzt das VRTX/Series-32000-System selber keinen der System-Interrupts und erfordert auch nicht, daß die Interrupt-Steuerung des Anwenders irgendeinem speziellen Muster entspricht. In den meisten Echtzeit-Systemen arbeiten die peripheren Einheiten mit der CPU mit Hilfe von Interrupts zusammen und der größte Teil der Systemaufrufe gilt der Interrupt-Steuerung. Das VRTX/Series-32000-System bietet zusätzlich fünf Systemaufrufe, die speziell für den Einsatz durch die Interrupt-Steuerung entwickelt wurden.

Wie *Tabelle 6* zeigt, setzt einer dieser Aufrufe den „Tick“-Zähler. Zwei weitere bieten eine einfache Möglichkeit, einzelne Zeichen zwischen dem VRTX-Zwischenspeicher und der Interrupt-Einheit auszutauschen. UI ENTER wird am Beginn des Steuerungsablaufs eingesetzt, um anzuzeigen, daß VRTX-Funktionen benutzt werden. UI EXIT wird im Normalfall am Ende einer Interrupt-Steuerung eingesetzt. Dieser Befehl bewirkt ein „Reschedule“ des VRTX/Series-32000-Systems, wenn die Interrupt-Service-Routine darin endet, daß eine Task mit höherer Priorität für die Bearbeitung bereit ist.

Erweiterungen

Die Funktionen des VRTX/Series-32000-Systems stellen einen Grundservice dar, der von den meisten Echtzeit-Applikationen gefordert wird. Zusätzlich zu diesen Basisfunktionen besteht aber häufig ein Bedarf für weitere, vom Anwender zu leistende Service-Funktionen. Im Interesse einer hohen Software-Zuverlässigkeit soll-

Tabelle 4

SC_GTIME	Get Time: obtains the current value of the clock counter.
SC_STIME	Set Time: sets the clock counter to a new value.
SC_TDELAY	Task Delay: suspends execution of the task for a specified number of clock ticks.

Tabelle 5

SC_GETC	Get Character: obtains the next character from the device buffer; if the buffer is empty the task is suspended until a character arrives.
SC_PUTC	Put Character: transmits a character to the device buffer; if the buffer is full the task is suspended until space for the character becomes available.
SC_WAITC	Wait For Special Character: suspends calling task until a particular character is received from the device.

Tabelle 6

UI_ENTER	Enter Interrupt Handler: notifies VRTX that an interrupt handler that uses VRTX services has been entered.
UI_TIMER	Announce Timer Interrupt: notifies VRTX that a tick has transpired.
UI_RXCHR	Post Received Character From Interrupt: transfers a character obtained from the device to the VRTX buffer without causing rescheduling..
UI_TXRDY	Post Transmit Ready From Interrupt: obtains a character from the VRTX character buffer so it can be transmitted to the device; does not cause rescheduling.
UI_EXIT	Exit From Interrupt: returns from interrupt and causes rescheduling if necessary.

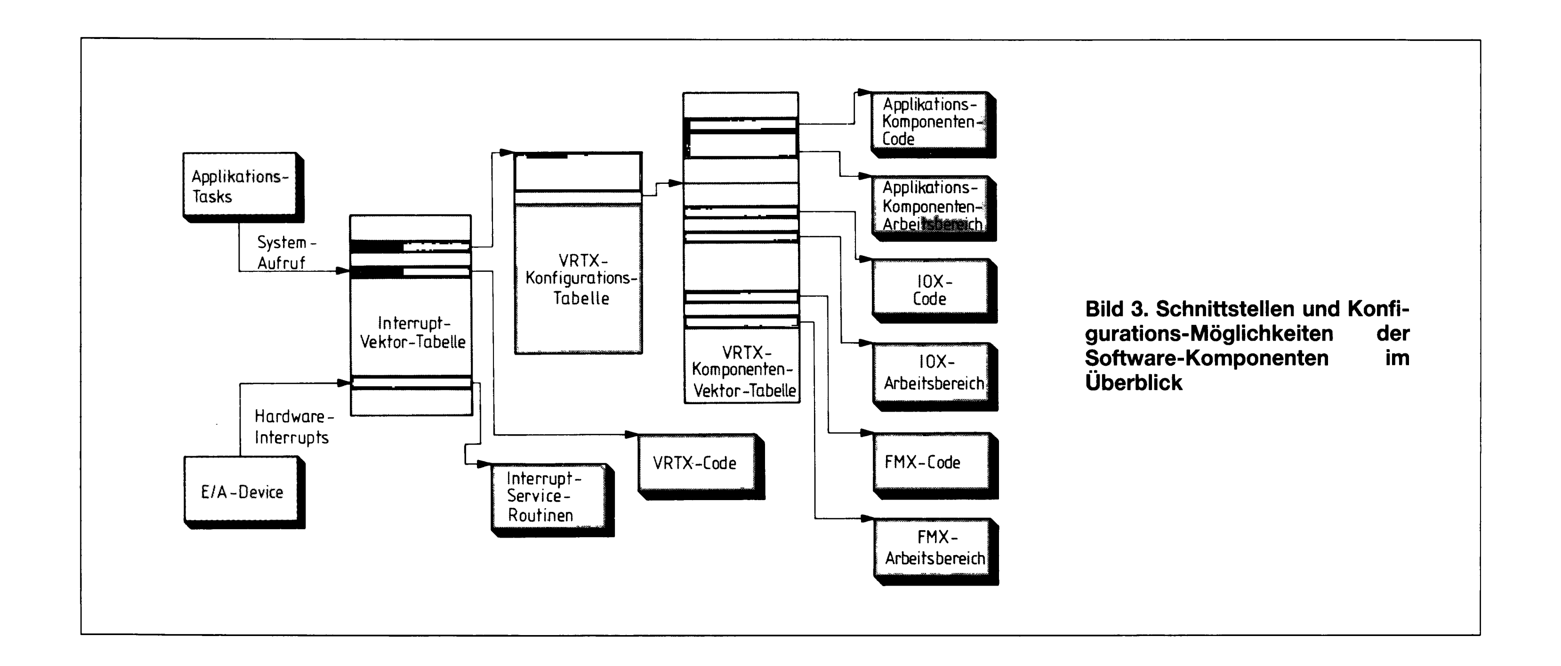


Bild 3. Schnittstellen und Konfigurations-Möglichkeiten der Software-Komponenten im Überblick

ten Anwender den VRTX/Series-32000-Code nicht verändern. Sie können ihn jedoch durch Hinzufügen von speziellen Routinen mittels dreier Software-„Hooks" anpassen, die für diese Zwecke zur Verfügung stehen.

Die drei „Hooks", bezeichnet als TCREATE, TDELETE und TSWAP werden dazu benutzt, vom Anwender zur Verfügung gestellte Applikations-Routinen aufzurufen, sofern welche vorhanden sind, und sie ermöglichen diesen Routinen den Zugriff auf einen Task-Control-Block (TCB). Der TCB enthält wichtige Daten, die das VRTX benötigt, um die Task und deren aktuellen Status zu definieren.

TCREATE wird automatisch aufgerufen, wenn das VRTX eine neue Task erzeugt. Wenn TCREATE aufgerufen wurde, so wird die Adresse des TCB der neuen Task für die Anwenderroutine verfügbar gemacht, die typischerweise eine TCB-Erweiterung erzeugt und initiiert. Die zahlreichen Anwendungsmöglichkeiten umfassen Leistungsmessungen und die Initialisierung von zusätzlicher Hardware wie z. B. Fließkomma-Prozessoren (FPUs). Das komplementäre „Hook", TDELETE, wird immer dann aufgerufen, wenn das VRTX im Begriff ist, eine Task zu tilgen.

Das dritte „Hook", TSWAP, wird aufgerufen, wenn das VRTX zwischen Tasks hin- und herschaltet. Die TSWAP-Routine hat Zugriff auf die TCBS der suspendierten Task und der Task, die zur Bearbeitung bereitsteht.

Konfiguration des VRTX/Series-32000-Systems

Das VRTX-System erfordert in bezug auf die Systemumgebung, in der es laufen soll, einige Voraussetzungen. Der VRTX-Code kann an jeder beliebigen Stelle im Speicher stehen, in einem ROM oder RAM, und es gibt verschiedene Schlüssel-Pointer und Operations-Parameter, die definiert sein müssen, bevor VRTX das erste Mal aufgerufen wird. Zusätzlich kann das VRTX-System zusammen mit einer Vielzahl anderer Software-Komponenten eingesetzt werden, sowohl mit Standard-Modulen wie das IOX/Series-32000 als auch vom Anwender erstellten Komponenten. Die Verbindungen zwischen den verschiedenen Software-Teilen müssen natürlich während der Systeminitialisierung geschaffen werden.

Die meisten Informationen, die die Betriebsumgebung des VRTX-Systems beschreiben, sind in einer kurzen Tabelle namens VRTX-Konfiguration-Table enthalten. Unter den Parametern, die darin zu finden sind, ist die Startadresse des VRTX-Arbeitsspeicher-Bereiches, die höchste Anzahl der Tasks und die Adressen aller installierten „Hook"-Routinen. Die zwei wesentlichen Betriebsparameter sind deshalb die Startadresse der Konfigurationstabelle und des aktuellen VRTX-Codes. Diese Adressen sind in der Interrupt-Dispatch-Tabelle des Systems abgelegt.

Zusätzlich zu der Konfigurationstabelle kann es auch eine VRTX-Component-Vektortabelle geben, wie *Bild 3* zeigt. Wenn ein Systemaufruf die Steuerung an das VRTX/Series-32000-Betriebssystem übergibt, so wird auch ein zweiteiliger Funktionscode mitgeschickt, der nicht nur die Operation spezifiziert, die ausgeführt werden soll, sondern auch den jeweiligen Software-Teil, der dafür zuständig ist. Dies kann das VRTX-System selber sein, ein damit verbundenes Modul wie IOX oder FMX oder ein vom Anwender geschriebenes Software-Modul.

Der Zweck der Component-Vektor-Tabelle ist es, dem VRTX-System die Steuerung der zugeordneten Software-Komponenten zu übergeben, sofern der Funktionscode anzeigt, daß dies erforderlich ist. Die Startadresse der Component-Vektor-Tabelle wird aus der VRTX-Konfigurationstabelle abgeleitet, und ein Index, der aus dem

Funktionscode abgeleitet ist, wird dann benutzt, um die Code- und Arbeitsspeicher-Adressen der Software-Komponenten festzulegen, die bearbeitet werden sollen. Das Protokoll zur Steuerung der Weiterleitung der Parameter zwischen den Software-Komponenten erlaubt darüber hinaus, daß das VRTX-System andere Software-Komponenten ebenfalls unterstützt. So macht zum Beispiel das IOX/Series-32000 einen extensiven Gebrauch von den VRTX-Intertask-Kommunikations-Möglichkeiten.

RESET

Initialisiere Speicher-Subsystem falls erforderlich

Interrupt-Vektor-Tabelle im ROM?

ja

nein

Schreibe Adressen der VRTX-Eingangspunkte und Konfigurationstabelle in Interrupt-Tabelle

Ausgabe: VRTX_INIT

Initialisierung der übrigen Hardware (Timer, E/A usw.)

Aufbau der Ausführungsumgebung mit Hilfe von VRTX-Aufrufen für Warteschlangen usw.

Ausgabe: VRTX_GO

Bild 4. Allgemeine Prozedur zur Konfiguration eines VRTX/Series-32000-Systems

Obwohl die detaillierte Konfiguration von der jeweiligen Applikation abhängig ist, demonstriert *Bild 4* in anschaulicher Weise die allgemeine Prozedur. Die Sequenz ist normalerweise Teil der Initialisierungsroutine, die als Antwort auf ein Reset-Signal abläuft. Falls das System einen virtuellen Speicher benutzt, so muß die MMU zuerst initialisiert werden. Als nächstes: falls die Interrupt-Dispatch-Tabelle nicht bereits in einem ROM codiert ist, so muß sie in ein RAM geladen werden. Das betrifft speziell die Zeiger für den VRTX-Code-Bereich und die Konfigurationstabelle.

Es gibt darüber hinaus verschiedene interne VRTX-Variable, die initialisiert werden müssen. Dies geschieht durch den Systemaufruf VRTX INIT. Wenn die Steuerung zurück an die Anwendungs-Initialisierungs-Routine geht, so besteht die nächste Aufgabe darin, Interrupt-Controller, Zeitgeber, I/O-Controller und alle anderen peripheren Einheiten zu initialisieren.

An diesem Punkt können dann die VRTX-System-Calls aufgerufen werden, um die Start-Ablaufkonfiguration zu definieren. Typische Aktivitäten in dieser Phase umfassen die Erzeugung von Tasks und Warteschlangen und das Festlegen von Speicherbereichen. Zum Schluß löst der VRTX GO-Systemaufruf, aus dem es jetzt kein Zurück mehr gibt, den Ablauf der Task mit der höchsten Priorität aus, die zur Verarbeitung ansteht.

Michael G. Holland, B. A

Für Echtzeit-Verarbeitung mit 32000-Prozessoren:

Echtzeit-Multitasking-Betriebssystem-Kern EXEC

Zahlreiche industrielle DV-Systemapplikationen erfordern die gleichzeitige Bearbeitung verschiedener Abläufe (Multitasking) in Echtzeit. Für die Mikroprozessoren der Serie 32000 ist ein Betriebssystem im Source-Code mit der Bezeichnung „EXEC" verfügbar, das eine kostengünstige Lösung für Multitasking-Echtzeit-Applikationen bringt. EXEC ist ein vom Anwender konfigurierbarer Betriebssystem-Kern, der in einem ROM speicherbar ist. Im folgenden wird dargestellt, wie ein solcher Kern aufgebaut ist und welches seine Funktionen sind.

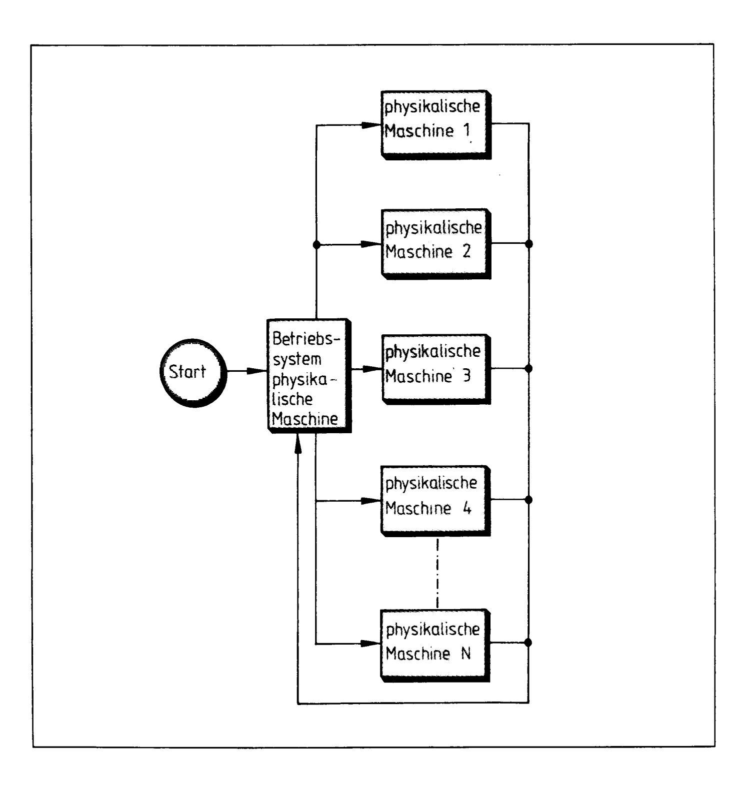
Kostengünstige Lösungen für spezielle Applikationen

EXEC umfaßt je einen Satz von Betriebsoperationen und Betriebsanweisungen, die den Ablauf von anwenderdefinierten Tasks und deren Interaktion mit dem Kern ermöglichen. Die Einbindung von EXEC in ein Betriebssystem erfordert die Erstellung von anwenderdefinierten Tasks, Kommunikations-Kanälen und Gerätetreibern, die alle zusammen die notwendigen Dienstleistungen erbringen. Diese kostengünstige Realisierung eines applikationsspezifischen Echtzeit-Betriebssystems wurde für solche Anwendungen entwickelt, bei denen der relativ geringe Auftragsumfang eine große Investition für ein Standard-Betriebssystem nicht rechtfertigen würde.

Der aktivste Teil des Systems

Der Kern eines Betriebssystems ist jener Teil mit den meisten Aktivitäten. Er ist deshalb in der Regel im Hauptspeicher resident. Die Funktionen eines Kerns verändern sich mit der Komplexität eines Systems, lassen sich jedoch folgendermaßen gliedern:
1. Ablaufsteuerung
2. Steuerung der Kommunikation zwischen dem Prozeß, den peripheren Einheiten und dem Betriebssystem
3. Ausnahme-Ablaufsteuerung (Interrupt und Trap-Handling)
4. Speicherverwaltung (Haupt- und Sekundärspeicher)

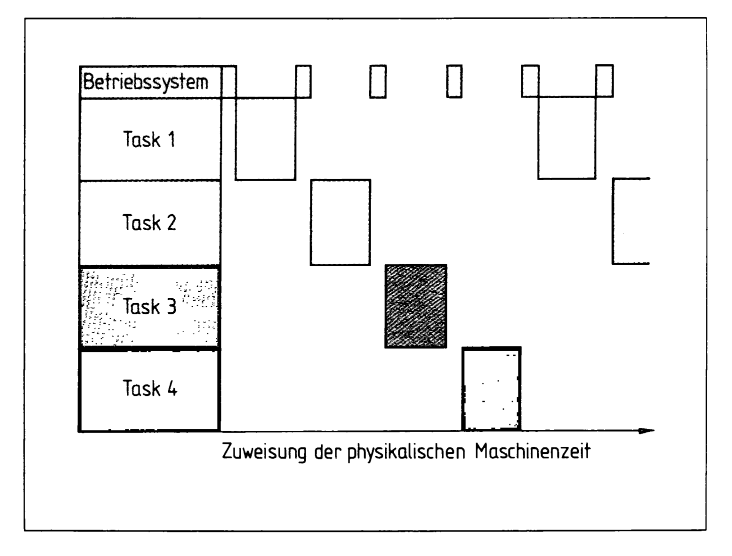
Weitere Systemfunktionen lassen sich als Teil in einem speziellen Kern berücksichtigen, abhängig jeweils von ihrer relativen Wichtigkeit oder der Häufigkeit ihres Einsatzes. Ein Kern ist der wesentliche Teil eines Betriebssystems, aber nicht alles. Er ist jedoch eine gute Basis, auf der ein leistungsfähiges Steuerprogramm, oder darüber hinaus ein komplexes Betriebssystem auf-



gebaut werden kann. EXEC ist am besten geeignet für Prozeßsteuerungs-Applikationen, zum Beispiel für anspruchsvolle Flüssigkeits-Meßvorgänge, Robotersteuerungen oder zur Steuerung von Umweltschutz-Einrichtungen, wo eine unmittelbare Reaktion von großer Bedeutung ist. In solch einer Applikation ist das Steuerungsprogramm genau festgelegt. Es ist nur wenig, wenn überhaupt, Interaktion mit dem Bediener erforderlich oder erwünscht.

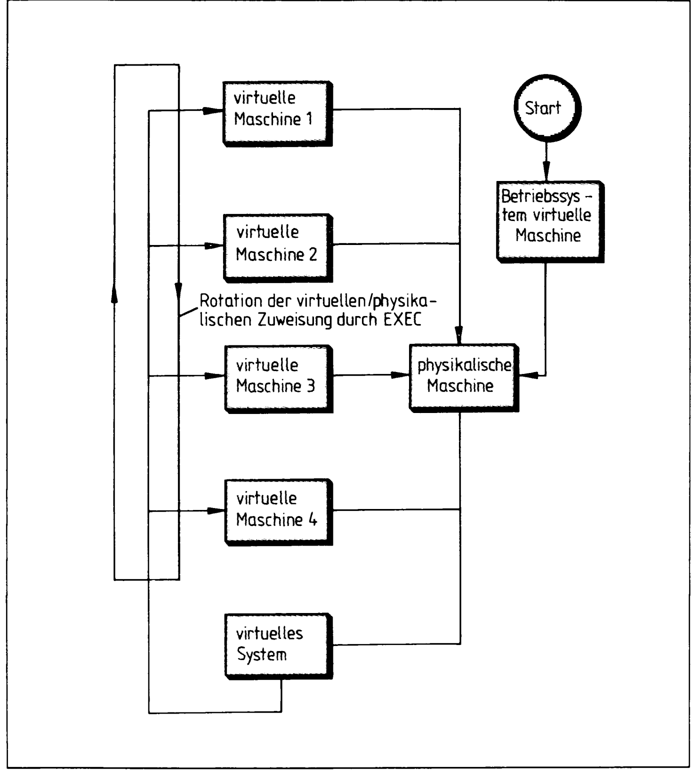
In einem Interrupt-gesteuerten System erzeugt eine externe Einheit eine Interrupt-Anforderung für den Prozessor, wenn sie eine Verbindung mit diesem benötigt. Der Prozessor sollte dann sehr schnell reagieren, um die angeforderte Leistung zu erbringen. Bei einem Echtzeit-System geht man noch einen Schritt weiter: Es ermöglicht die Programmierung eines Taktsignals, das zur Erzeugung von Interrupts zu den kritischen Zeitpunkten eines Ablaufes dient, so daß zusammengehörige Aktionen, wie die Erfassung eines Meßwertes, der Austausch einer Information oder das Schalten eines Ventils, stattfinden können. Wenn dieses Taktsignal darüber hinaus zur Umschaltung zwischen den Tasks benutzt wird, so ist damit die Grundlage für ein Echtzeit-Multitasking-System geschaffen.

Die Effektivität eines Mikroprozessors wird verringert, wenn er auch periphere Operationen wie das Ausdrukken von Texten oder das Plotten von Grafiken ausführen muß, die viel Zeit in Anspruch nehmen. Man muß dabei berücksichtigen, daß ein Prozessor im Durchschnitt etwa 1 Million Instruktionen pro Sekunde ausführen kann und ein Drucker etwa 10 ms benötigt, um eine Zeichenkette auszugeben. Der Prozessor ist somit untätig, während er auf den Drucker warten muß. In dieser Zeit, von 10 ms, könnte der Prozessor rund 10 000 Instruktionen bearbeitet haben.



Für dieses Problem gibt es zwei grundsätzliche Lösungen. Eine besteht darin, daß jede Aufgabe einem eigenen Prozessor zugewiesen wird, was also ein „Multiprozessor“-System bedeutet, in dem ein Prozessor die Rolle des „System-Managers“ übernimmt (*Bild 1*). Diese Lösung erfordert einen hohen Hardware-Aufwand und ist in bezug auf die Hard- und Software-Entwicklung teuer. Die Lösung ist aber auch hinsichtlich der Ausnutzung der Prozessoren ineffizient, weil jeder Prozessor inaktiv ist, wenn seine Task-Operationen mit der peripheren Hardware ausführt.

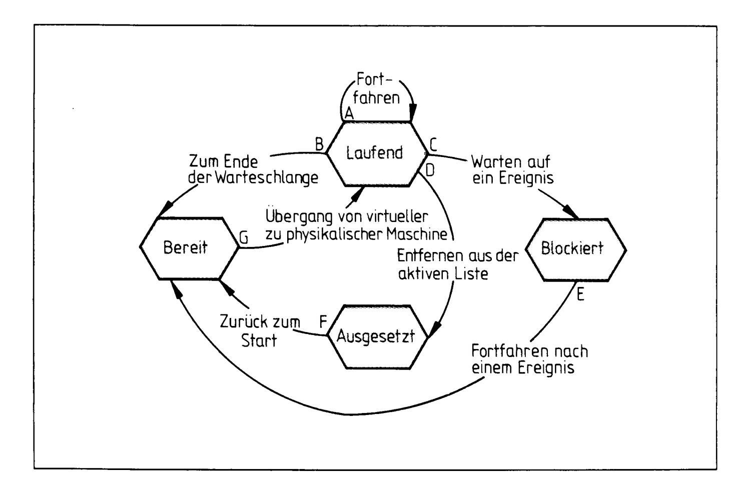
Die zweite, kostengünstigere Lösung besteht darin, die Prozessor-Zeit effektiver zu nutzen. Es ist die „Multi-Processing“-Lösung (*Bild 2*). In einer Multi-Processing- oder Multitasking-Applikation gibt es den positiven Effekt, daß die Prozessor-Zeit zwischen verschiedenen Abläufen geteilt wird, so daß jede Aufgabe in einer vordefinierten Zeitperiode abläuft. Diese Art der Verarbeitung nutzt der Prozessor in effizienterer Weise, weil hierdurch die tatsächliche „Arbeitszeit“ des Prozessors verlängert wird und alle aktiven Tasks so unterstützt werden, als wenn sie parallel abliefen (*Bild 3*).



Prozeß-Zustände

In einem EXEC-Kanal kann ein laufendes Programm, in diesem Beitrag auch als Prozeß oder Task bezeichnet, sich in einem der vier folgenden Zustände befinden:

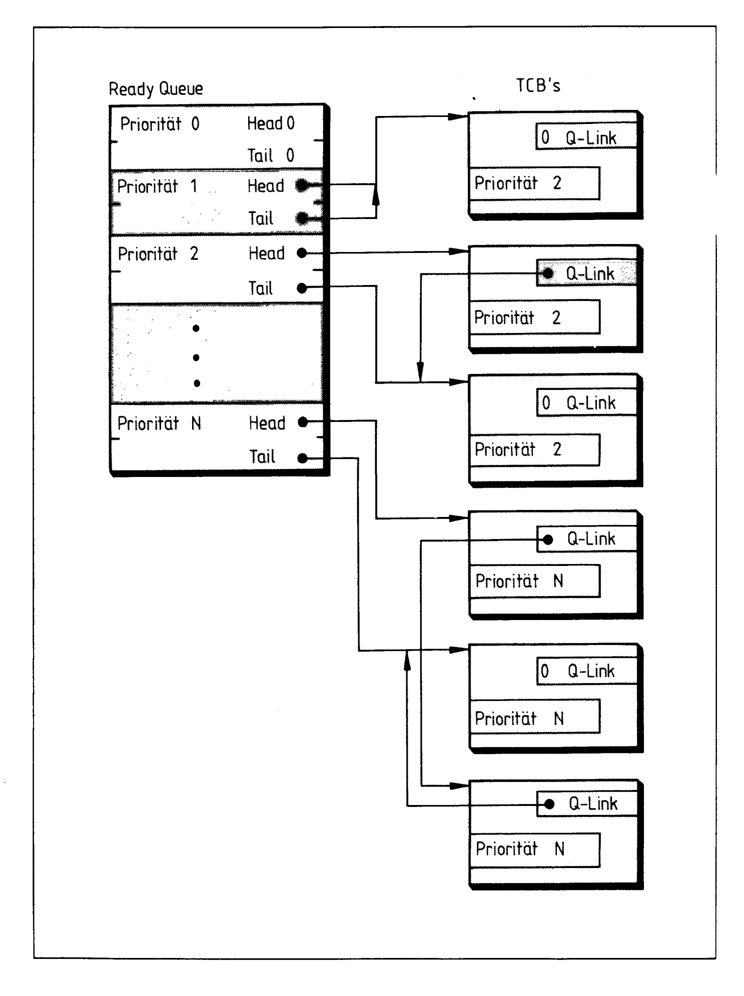
1. gerade ablaufend (running)
2. bereit (ready)
3. blockiert (blocked)
4. ausgesetzt (suspended)



Die ersten drei Zusätze sind „aktiv", so daß die Tasks jeweils CPU-Zeiten beanspruchen. Der vierte Zustand ist „inaktiv" (dorment) und beansprucht keine CPU-Zeit. *Bild 4* zeigt ein Zustandsdiagramm [1, 2]. Zur selben Zeit kann jeweils nur eine Task laufen. Wenn eine Task die erste in einer Liste oder „Warteschlange" von „Ready-Tasks" ist, so wird sie in den Ablauf übernommen und ihr Eintrag aus der Liste bzw. der „Warteschlange" entfernt. Eine „Ready-Queue" ist eine „First-in-First-out"-Liste von Task-Zeigern (*Bild 5*). Jede Task-Prioritätsebene hat einen entsprechenden Eintrag in der „Ready-Queue". Falls es bereits „Ready-Tasks" mit der entsprechenden Priorität gibt, hat auch der Eintrag entsprechende Zeiger. Sofern keine Tasks auf der entsprechenden Prioritätsebene bereit sind, enthält der Eintrag keine Zeiger. Die Zeiger sind „Task-Steuer-Block-Plätze" (TCB). Ein TCB kennzeichnet eine virtuelle Maschine, die, in anderen Worten, den Zustand eines Prozessor-Registers für eine Task darstellt, wenn sie sich im Ablaufzustand befindet. Die TCBs bilden eine Verbindungsliste aller aktiven Prozesse. Tasks auf derselben Prioritätsebene, die vom Anwender per Parameter festgelegt wird, bilden ebenfalls eine Verbindungsliste. Die „Ready-Queue"-Eintragungen zeigen für jede Priorität auf den ersten und letzten TCB. Es ist die Aufgabe des Kerns, diese Listen zu verwalten.

Nach der Systeminitialisierung nimmt der Kern seine Arbeitsfunktionen erst dann wahr, wenn ein „Exception" auftritt: zum Beispiel, falls eine Task die ihr zugewiesene Zeit überschreitet, eine Task mit höherer Priorität die CPU in Beschlag nimmt, wenn wegen des Wartens auf ein Ereignis, wie beispielsweise der Empfang eines Signals, eine Blockierung stattfindet oder wenn eine Service-Routine aufgerufen wird. Die Interrupt-Controller-Einheit NS32202 kann ein Echtzeit-Taktsignal liefern. Sie hat zwei Zähler von denen einer für diese Funktion einsetzbar ist. Der Anwender legt die Interrupt-Periode, die auch als „Time-Slice" bezeichnet wird, in Millisekunden fest. Darüber hinaus definiert er die Zeitspanne, die für die Abarbeitung der Tasks zur Verfügung steht, wenn diese den Ablaufstatus erreichen, was bei EXEC von den jeweiligen Prioritäten abhängig ist. Die maximale Zeitspanne, die für den Ablauf einer Task zur Verfügung steht, ist ein Vielfaches der Interrupt-Periode bzw. des „Time-Slice". Ist zum Beispiel die Interrupt-Periode mit 10 ms festgelegt und die Task-Periode mit dem zehnfachen Wert, so sind jeder Task 100 ms der CPU-Zeit zugewiesen (*Bild 6*). Die jeweils laufende Task wird unterbrochen, wenn die ihr zur Verfügung stehende CPU-Zeit abgelaufen ist. Der Kern muß nun seine Steuerfunktionen wahrnehmen und entscheiden, ob die abgebrochene Task weiterlaufen kann oder ob eine Task mit höherer Priorität bereitsteht, die zunächst einmal abgearbeitet werden soll. Die laufende Task kann eine „Supervisor-Call-Instruction" ausführen, um den Kern „auf sich aufmerksam zu machen". Wenn er diese „Exception" bearbeitet hat, muß der Kern von neuem entscheiden, ob die Task weiterlaufen kann oder nicht.

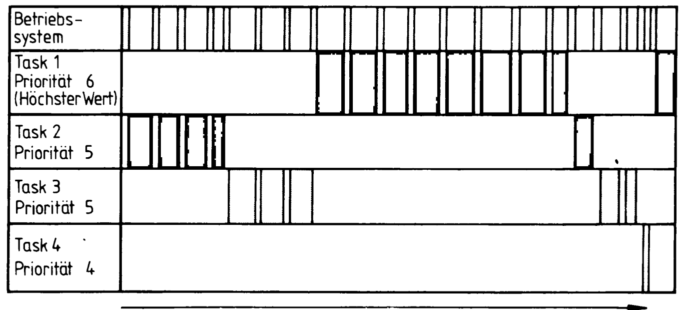
Falls die Task eine Operation mit einer peripheren Einheit erfordert, so kann sie so lange blockiert werden, bis die jeweilige Operation vollständig vorbereitet ist. Falls die CPU-Zeit für eine Task noch nicht abgelaufen ist und es keine Anforderung durch eine Task höherer Priorität gibt, so kann der Ablauf bis zum Ende der nächsten Interrupt-Periode (Time-Slice) fortgesetzt werden, nachdem der Kern die jeweiligen Routinearbeiten durchgeführt hat, wie zum Beispiel die Aktualisierung

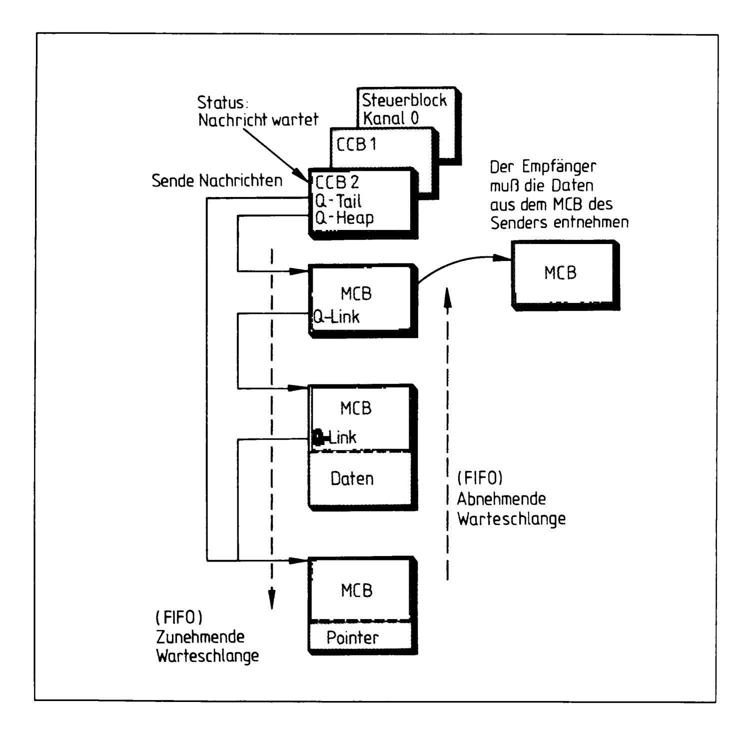


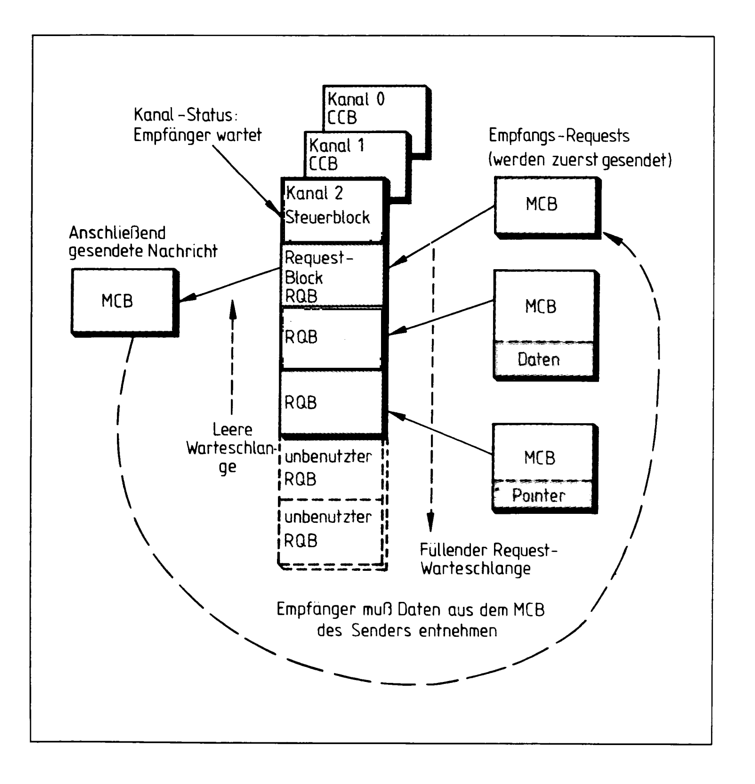
des Task-Status. Eine Task, die ihre zur Verfügung stehende Zeit „verbraucht“ hat, wird automatisch in den „Ready“-Status versetzt und an das Ende ihrer „Priority-Ready-Queue“ gehängt, so daß sie für eine weitere Bearbeitung „vorgemerkt“ ist. Eine Task wird blockiert, wenn sie auf ein Ereignis wartet. Falls das Ereignis dann eintritt, wird die Task in den „Ready“-Zustand versetzt und an das Ende der jeweiligen „Ready-Queue“ gesetzt. Eine Task kann von einer anderen Task oder von sich selbst abgesetzt werden. Ihr „Stack“-Bereich ist dann frei für die Benutzung durch andere Tasks und ihre TCB wird aus der „ACTIVE-Queue“ entfernt. Die Task läßt sich auch dadurch fortführen, daß sie durch eine andere Task wieder gestartet wird. Die Fortführung der Task setzt voraus, daß die Bedingungen hergestellt sind, die am Beginn der Bearbeitung vorhanden waren. So wird der TCB an das Ende der betreffenden „Ready-Queue“ gehängt und mit der „ACITVE-Task-List“ verbunden.

Inter-Task-Kommunikation

Die Kommunikation zwischen Tasks und ebenso zwischen Task und der Peripherie wird mit Hilfe der Software- und Interrupt-Kanäle abgewickelt. Die Kanäle können beim Programmablauf oder während der Initialisierungs-Konfiguration bezeichnet werden. Die Zahl der vorhandenen Kanäle ist ein vom Anwender festlegbarer Parameter. Ein Kanal ist in erster Linie ein „Channel-Control-Block“ (CCB), den man als virtuellen Kommunikationskanal betrachten kann. Alle zur Kommunikations-Transaktion erforderlichen Informationen sind im CCB gespeichert. Wenn ein Prozeß-Zugriff auf einen Kanal erfolgt, so erfolgt auch ein Zugriff auf einen „Message Pointer“ oder ein „Message-Pointer“ wird empfangen, je nachdem, ob eine Empfangs- oder Sende-Anforderung vorliegt. Die Nachricht (Message) ist Bestandteil eines „Message-Control-Block“ (MCB). Die Task, die eine Message empfängt, erhält Zugriff auf den MCB. Sie muß dann die Daten extrahieren, die in dem MCB oder in einem Puffer gespeichert sind, dessen Adresse in einem Datenfeld des MCB übermittelt wird. Einem Kanal können beliebig viele Tasks zugeordnet sein. Ein Kommunikationsprotokoll, falls erforderlich, muß vom Anwender definiert und von der sendenden und empfangenden Task erkannt werden. Falls an den Kern eine Sendeanforderung geht (*Bild 7*), wenn der Kanal nicht im Status „Receiver waiting“ ist, weist dieser dem MCB den definierten Kanal zu. Weitere Sende-Aufrufe bewirken, daß eine MCB-Warteschlange aufgebaut wird. Eine Nachricht wird empfangen, indem ein Leseaufruf (*Bild 8*) erfolgt. Wenn der Kanalstatus nicht „Warten auf Message“ ist, wird ein „Request-Block“ (RQB) dem Kanal zugeordnet. Wurde eine Nachricht abgeschickt, wäh-

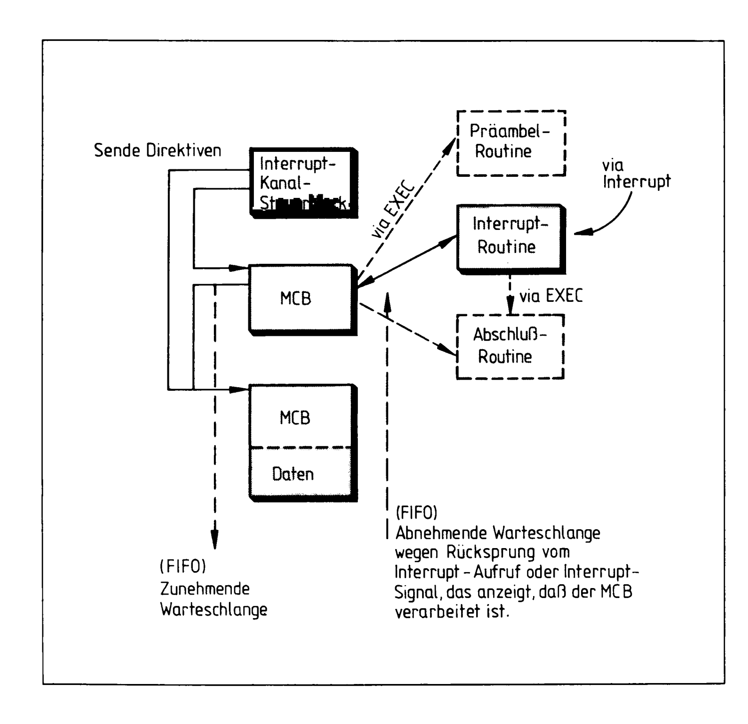






rend der Status des Kanals „Receiver waiting“ war, so wird ein Pointer des Sender-MCB zum Empfänger geleitet und der RQB aus dem Kanal entfernt. In gleicher Weise geschieht es, wenn eine Empfangs-Aufforderung erzeugt wurde, aber schon eine Nachricht wartet. Nun wird ein Pointer des ersten MCB in der Schlange zum Empfänger geleitet und der MCB aus der Warteschlange genommen.

Ein Interrupt-Kanal funktioniert auf ähnliche Weise (*Bild* 9), in dem ein Interrupt-Kanal-Steuerblock erzeugt wird, dessen Eintragungen sich nur wenig von einem Software-CCB unterscheiden. Ein Interrupt-Kanal benutzt einen Device-Treiber, um mit der Hardware zu kommunizieren. Zu einem Treiber können eine Einleitungsprozedur und eine Abschlußprozedur gehören. Die Einleitung läßt sich zur Initialisierung einer Einheit, zur Datenmanipulation oder zu beiden benutzen. Die Abschlußprozedur kann man zur Deinitialisierung einer Einheit oder zum Abschluß eines Datenverarbeitungs-Vorganges am Ende einer Kommunikationsprozedur benutzen.



Der Device-Treiber wickelt den Datentransfer zwischen der Hardware und dem Datenpuffer ab. Nach der Zuordnung des angeforderten Kanals wird ein MCB für einen EXEC-Sende-Systemaufruf initialisiert. Das Format der vom Benutzer definierten Teile des MCBs ist für den Device-Treiber bestimmend, weil er aus den darin enthaltenen Informationen die Aktionen interpretiert, welche nun stattfinden sollen. Der MCB befindet sich in der Warteschlange des Kanals und die Einleitungsroutine des Device-Treibers kann aufgerufen werden, wenn der MCB an die erste Stelle der Schlange gerückt ist, um die Einheit zu aktivieren oder die Steuerparameter an den „Device-Handler“ weiterzuleiten. Sobald der Interrupt-Service abgeschlossen ist, wird der MCB aus der Warteschlange entfernt, was durch einen „Interrupt-Exit“ oder durch Signalanweisungen ausgelöst wird. Eine Sende-Anweisung wird auch dazu genutzt, um Daten auf einem Interrupt-Kanal zu empfangen. Die Richtung des Datenflusses durch den Interrupt-Kanal wird durch das MCB-Protokoll gesteuert, das vom Anwender definiert ist. Einem Interrupt-Kanal können zahlreiche Tasks zugeordnet werden, aber zu jedem Kanal gehört nur ein Device-Treiber. Ein Device-Treiber kann jedoch mit verschiedenen Interrupt-Kanälen verbunden sein. Beispielsweise können die Empfangs- und Sende-Signale für eine serielle Interface-Einheit auf zwei Interrupt-Eingänge der Interrupt-Steuereinheit gelegt und durch denselben Device-Treiber unterstützt werden.

Hauptspeicher-Verwaltung

Auch die Einsatzverwaltung von freien Bereichen des Hauptspeichers (RAM) ist eine Funktion des Kerns. Jeder Task kann ein zur Abspeicherung von Variablen nicht näher definierter Speicherbereich zugewiesen werden. Wenn EXEC aufgerufen wird, um eine Task zu erstellen, so wird die Größe des zugeordneten Stacks vom Benutzer definiert. Die verbleibenden, nicht zugeordneten RAM-Bereiche sind für dynamische Speichervorgänge verfügbar. Eine Task kann aus diesem freien Pool zusätzlichen Speicherraum anfordern oder nicht benutzte Speicherkapazität nach der Benutzung freigeben. Verschiedene Tasks können auch bestimmte Bereiche untereinander teilen (Partitions). Diese Partitions lassen sich in Blöcke einer definierten Größe segmentieren und bestimmten Tasks zuordnen oder von diesen zur anderen Benutzung freigeben. EXEC übernimmt die Steuerung des Pools freier Speicherplätze, wobei Pointer auf die höchsten und niedrigsten freien Speicherplätze zeigen. Prozeduren regeln die Zuordnung und die Freigabe von Partitions. In der Initialisierungsphase wird eine Liste von Speicherpartitions erstellt (MPTs). Zusätzlich gibt es eine Liste der „in Gebrauch“ befindli-



Michael G. Holland B.A. Cert Ed. stammt aus Hereford, England. Er studierte Mathematik sowie Elektrotechnik und ließ sich anschließend zum Gewerbelehrer ausbilden. In den letzten Jahren arbeitete er bei verschiedenen ATE- und Halbleiterherstellern als Ausbildungsleiter oder Applikations-Manager. Seit 2½ Jahren ist er bei National Semiconductor in Fürstenfeldbruck europaweit für das Mikrosystem-Training verantwortlich. In seiner Freizeit beschäftigt er sich mit Lesen, Wandern oder Gartenarbeit.

chen MPT, die von EXEC geführt wird. Es liegt in der Verantwortung des Benutzers, die Zuordnung und Freigabe von Blöcken von den ihnen zugewiesenen Partitions zu handhaben. Eine allgemein übliche Anwendung dafür ist die Verwendung von Blöcken für Message-MCBs. Eine sendende Task ordnet jeder empfangenden Task einen Block zu, den der Empfänger wieder freigibt, wenn er die Daten verarbeitet hat, die der Sender darin abgelegt hat. Falls keine weiteren Blocks mehr zugewiesen werden können, muß der Sender so lange innehalten, bis der Empfänger die Verarbeitung abgeschlossen hat und einige Blöcke für die weitere Benutzung frei werden. Dieses Verfahren verhindert einen Kommunikationsdaten-Überlauf, bei dem Informationen durch das Fehlen von Speicherplatz für die MCB-Zuordnung verlorengehen.

Programm-Ablaufdiagramm

Das EXEC-Flußdiagramm (*Bild 10*), zeigt die wesentlichen Merkmale der Arbeitsweise dieses Programms. Nach dem Einschalten oder einem Reset schaltet die Steuerung zuerst zum „RESET-Entry-Point" des CONFIG-Programm-Moduls, der alle vom Benutzer festgelegten System-Parameter beinhaltet. Der Aufruf dieses Moduls durch EXEC erfolgt über eine Programm-Generaladresse. Der „Konfigurator" enthält darüber hinaus die Prozeduren „INIT16" und „COLD16", mehr darüber im weiteren Verlauf dieses Beitrags. Einige Prozessor-Register werden zuerst initialisiert, dann geht die Steuerung an EXEC über. Danach werden die folgenden Operationen durch EXEC ausgeführt:

1. Löschen des Generaladressenspeichers
2. Aufbau und Initialisierung der vom Benutzer definierten „Exception-Vector-List" (Dispatch-Table)
3. Fixierung (in EXEC-Globals) der vom Anwender in CONFIG festgelegten Grenzen für den freien Speicherbereich, Ereignisse und vektorisierte Interrupts
4. Erstellung einer Software- und Interrupt-Kanaltabelle, gefolgt von einer Ready-Warteschlangen-Tabelle mit einem Eintrag für jede Prioritätenebene
5. Erstellung einer Speicher-Partition-Tabelle
6. Setzen der Zeiger für die freien Speicherstellen
7. Initialisierung des Echtzeit-Taktsignals.

Start

Initialisiere:
Register
Tabellen
Grenzen
Pointer
Speicher

INIT 16
Aufbau der Software-kanäle, Tasks und Speicher-Partitions

COLD16
Initialisieren der NS 32202 Interrupt-Steuereinheit, Ablauf der Device-Treiber-Initialisierungs-Routinen

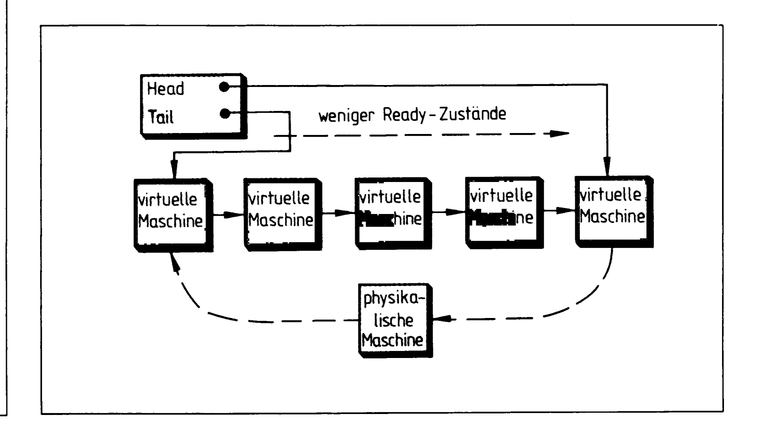
Dispatcher
Aktualisieren des virtuellen Maschinen-status

Ablauf der Task auf physikalischer Maschine

Exception-Service-Routine

Damit ist die Initialisierung von EXEC vollständig. EXEC ruft nun INIT16 auf, dessen Funktion es ist, am Anfang Software-Kanäle, Tasks und Speicherbereiche aufzubauen. Danach ruft EXEC COLD16 auf, das die Interrupt-Steuereinheit und periphere Hardware initialisiert sowie die Interrupt-Kanäle definiert. EXEC übernimmt daraufhin wieder die Steuerung und verzweigt zur Task-Dispatcher-Routine. Diese hat die Funktion, die mit der höchsten Priorität bereitstehende Task in den Ablaufstatus zu versetzen. Falls das geschehen ist, übernimmt die Task die Steuerung des Prozessors. Nachdem die jeweilige Operation abgeschlossen ist, kann eine Task in einer Schleife laufen oder sich selbst abschalten, was normalerweise der Fall ist. Dieselbe Task kann auch eine andere Task aufrufen, die unterbrochen oder zurückgestellt war. Deshalb muß sie immer aktiv bleiben, damit zurückgestellte Tasks aktiviert werden können.

EXEC übernimmt nach einer Exception, die durch eine normale Unterbrechung, einen Interrupt oder durch das Echtzeit-Taktsignal verursacht wurde, wieder die Steuerung. Der Dispatcher hat dann die Funktion, den Ablauf der nächsten Task zu steuern, wobei das eine Statusveränderung von der virtuellen zur physikalischen Maschine mit sich bringt.



Task-Zuordnung und -Priorisierung

Das in EXEC angewandte Task-Zuordnungsverfahren (*Bild 11*), „Round Robin“ genannt, regelt die Prioritätensteuerung einer Task in der Weise, daß eine laufende Task, die in den virtuellen Maschinenstatus zurückkehrt, am Ende der Warteschlange plaziert wird (Priority-Ready-Queue). Jeder Task, die den Ablaufstatus erreicht, wird die vom Anwender festgelegte, maximal zur Verfügung stehende CPU-Zeit zugeordnet. Natürlich kann eine Task auch auf die Inanspruchnahme der ihr zustehenden CPU-Zeit „verzichten“ und den Prozessor jederzeit wieder freigeben. Dieses Verfahren garantiert eine möglichst kurze Reaktionszeit in bezug auf Tasks höherer Priorität in einem Multitasking-System. Der erforderliche Aufwand für das Umschalten der Tasks ist sehr klein, wenn die Tasks im Hauptspeicher gehalten werden und ein effizienter Context-Umschaltmechanismus zum Einsatz kommt [2].

Die Zahl der Prioritätsebenen wird vom Anwender festgelegt. Die höchste Ebene hat auch die höchste Prioritätsziffer. Falls eine Task höherer Priorität, als die jeweils laufende, zur Verarbeitung bereitsteht, so hat sie ein „Vorrecht“ und die Verarbeitung der ablaufenden Task wird beendet. Die neue Task wird dann in den Ablaufstatus versetzt und die alte an das Ende der Prioritätsebenen-Schlange gesetzt. Solange wie es zur Verarbeitung bereitstehende Tasks auf der höchsten Ebene gibt, erlangt keine Task minderer Priorität Verfügung über den Prozessor. Es ist eine Angelegenheit der sorgfältigen Systementwicklung, sicherzustellen, daß alle Tasks adäquat unterstützt werden. Tasks können die Prioritätsebene wechseln, aber nicht über die Ebene hinaus, die ursprünglich in ihrer TCB definiert ist. Eine Task, die auf eine andere Prioritätsebene gewechselt ist und sich in der Warteschlange befindet, wird, wenn sie wieder in den Ablaufstatus versetzt wird, die neu definierte Priorität einnehmen. Eine blockierte Task nimmt, während sie mit einer neuen Priorität abläuft, ihre ursprünglich im TCB definierte Priorität wieder an, wenn sie in den „Ready„“-Status geht.

Es ist eine Sache der sorgfältigen Systemabstimmung, eine optimale Unterstützung für eine bestimmte Applikation und Prozessorbelastung zu erreichen. Interrupt-Prozeduren sollten ebenso wie Tasks höherer Priorität kurz gehalten werden. Dafür bietet sich die Technik an, eine Task auf eine niedrigere Prioritätsebene zu schalten, sobald alle wesentlichen Operationen beendet sind. Das Zeitfenster für die Verarbeitung und die Task-Periode müssen ebenfalls so gewählt werden, daß ein Optimum an Unterstützung und ein Minimum an Umschalt-Aufwand entsteht. Einige Experimente unter extremen, aber realistischen Arbeitsbelastungen werden gute Anhaltspunkte für vernünftige Werte dieser Parameter geben. Macht man die Prozessor-Zugriffszeit zu kurz, so tritt ein Effekt auf, der ähnlich ist wie bei der schlecht funktionierenden Speicherverwaltung: es wird mehr Zeit für das „Swapping“ benötigt als für die tatsächliche Verarbeitung zur Verfügung steht. Macht man die Prozessor-Zugriffszeit zu lang, so führt das dazu, daß umfangreiche Tasks die Prozessorzeit über Gebühr in Anspruch nehmen. Es gibt keine allgemein gültige Lösung für dieses Problem. Jeder Einzelfall erfordert eine gute Strategie und eine optimale Abstimmung der verschiedenen System-Parameter für eine bestmögliche Lösung.

EXEC-Anweisungen

Ein schneller Überblick über die EXEC-Anweisungen gibt einen Eindruck von den Möglichkeiten, die dieser Kern bietet. Die Anweisungen lassen sich in folgende Kategorien unterteilen [3]:

Kommunikation und Ereignis-Steuerung

XSEND	Senden einer Nachricht und Fortführen der Verarbeitung
XSENDB	Senden einer Nachricht und Blockierung bis zum vollständigen Empfang
XSENDW	Senden einer Nachricht und Blockieren bis zum Auftreten eines Signals
XRECV	Verknüpfen eines Ereignisses mit dem Empfang einer Nachricht
XRECVW	Blockieren einer Task, bis eine Nachricht verfügbar ist
XRECVT	Empfange Nachricht, falls verfügbar, und fahre fort
XSIGNL	Signalisieren der Vollständigkeit einer Nachrichtenverarbeitung
XWAITE	Blockieren einer Task bis zum Auftreten eines Ereignisses
XTSTEV	Prüfen auf ein bestimmtes Ereignis und Fortfahren in der Verarbeitung

Task-Steuerung

XTSKBD	Aufbau und Zuordnung einer neuen Task
XGTSKP	Hole Task-Paramter
XGTPRI	Hole Task-Priorität
XSTPRI	Setze Task-Priorität
XSUSPD	Unterbreche eine Task
XRESUM	Wiederaufnahme einer Task

Kanal-Steuerung

XBLDSC	Baue Software-Kanal auf
XBLDIC	Baue Interrupt-Kanal auf
XCANCHN	Lösche Kanal
XATTCHN	Weise einen Kanal zu

Zeitsteuerung

XMRKT	Verknüpfe ein Ereignis mit einer Zeitperiode
XMRKTW	Blockiere Task bis Zeitperiode abgelaufen ist
XCMRKT	Lösche festgelegte Zeit (falls möglich)
XGTIMD	Hole Tageszeit
XSTIMD	Setze aktuelle Tageszeit
XMRKTOD	Verknüpfe Ereignis mit Tageszeit

Dynamische Speichersteuerung

XPCREATE	Erzeuge eine Speicher-Unterteilung
XPRELS	Freigabe einer Speicher-Unterteilung
XPATACH	Füge zu einer Speicher-Unterteilung hinzu
XALLOC	Weise Speicherblöcke aus einer Partition zu
XDALOC	Retourniere Blöcke zu einer Partition

Systemsteuerung

XHALT	Halt des Kerns

Exception-Abwicklung

XDTRAP	Deklariere Trap-Exception-Abwicklung

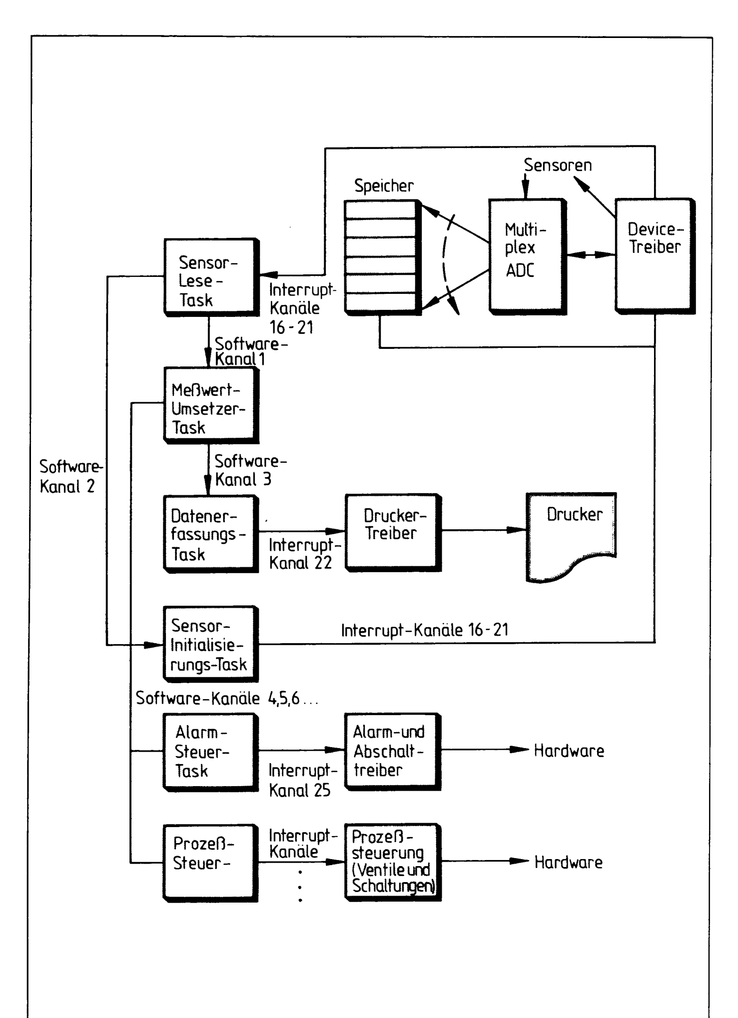
Interrupt-Service

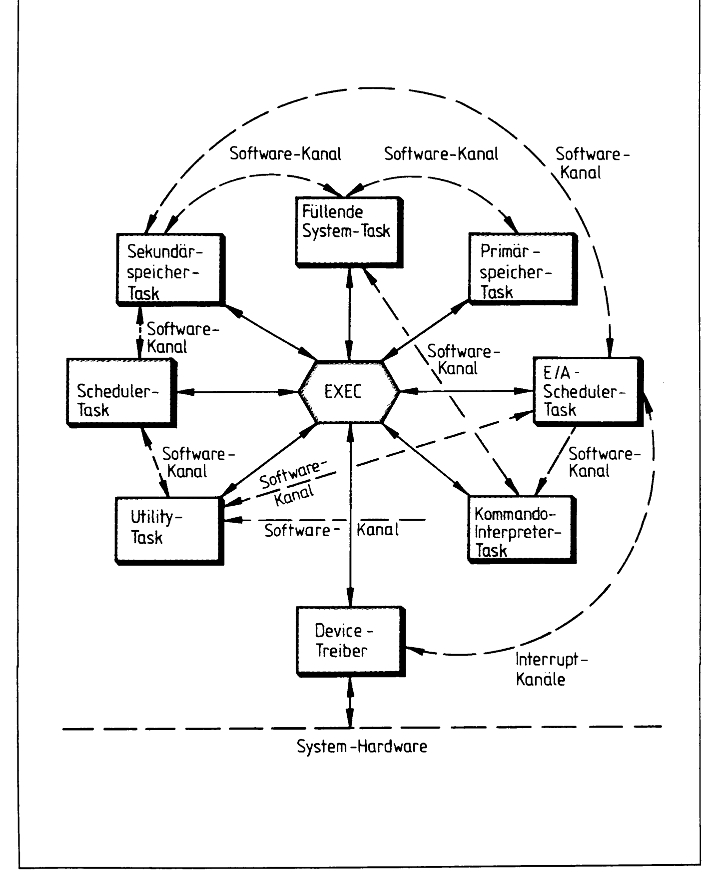
XXINTEX	Interrupt-Handler-Exit
XXPOSTE	Post-Event
XXACTVP	Aktiviere eine Prozedur nach Ablauf einer Zeit
XXSIGNL	Signalisiere Beendigung eines Aufrufes

Es ist nicht schwierig, dem Satz von Standard-Anweisungen weitere hinzuzufügen. Dies sollte man allerdings nur in Betracht ziehen, wenn eine wesentliche Kern-Funktion für eine spezielle Applikation nicht zur Verfügung steht.

Applikationen

Eine Applikation von EXEC in einem Prozeß-Steuersystem kann so aussehen, wie in *Bild 12* dargestellt. In diesem Falle geht es um ein anspruchsvolles Prozeß-Meßsystem. Hier dient eine Task der Erfassung digitaler Meßwerte über einen Interrupt-Kanal, der an einen Device-Treiber angeschlossen ist, welcher wiederum einen Multiplex-Analog-Digital-Umsetzer ansteuert. Der





Analog-Digital-Umsetzer wird von verschiedenen Sensoren mit Eingangssignalen versorgt. Die Steuerung dafür besorgt der Device-Treiber, der auch für die notwendige Synchronisierung sorgt. Die gemessenen Werte werden über einen Software-Kanal weitergeleitet und gelangen so zur Meßwert-Konvertierungs-Task, wo sie dann verarbeitet werden, um die erforderlichen Informationen bereitzustellen. Diese Informationen werden nun zur Datenerfassungs-Task, Prozeß-Steuer-Task und, falls Grenzwerte überschritten worden sind, zur Alarm- und System-Abschalt-Task über weitere Software- und Interrupt-Kanäle geleitet. Die Darstellung zeigt deshalb auch eher ein Kommunikations-Flußdiagramm als einen Programm-Ablaufplan. Die aktiven Tasks stehen sozusagen gegeneinander in Konkurrenz um den Zugriff auf die System-Ressourcen.

Bild 13 zeigt, wie EXEC erweitert werden kann, um die Leistungsfähigkeit des Systems zu steigern. Der Zeitaufwand für die Software-Entwicklung auf Anwenderseite hängt naturgemäß von der Systemkomplexität ab. Die jeweiligen Kosten müssen zu denen in Relation gesetzt werden, die beim Kauf eines Standard-Systems anfallen. Es kann durchaus der Fall sein, daß ein kleineres, kundenspezifisches System alle Anforderungen erfüllt. In diesem Falle ist EXEC eine gute Ausgangsbasis.

Literatur

[1] *Minsky, M.*: Computation. Finite and infinite machines. Prentice-Hall International, Inc., London (A mathematical treatment of state machines).

[2] *Deitel, M. H.: An Introduction to Operating Systems. (Revised First Edition). Addison-Wesley Publishing Company.*

[3] *Series 32000 EXEC: Romable, Real Time Multi-tasking Executive. Firmenschrift von National Semiconductor.*

Das „Tiny Development System" TDS

TDS ist ein Software-Paket für den Betrieb von Einplatinen-Systemen mit den Prozessoren der Serie 32000. Die dem Anwender damit gebotenen Dienstprogramme sind ein Text-Editor, ein Assembler und ein symbolischer Debugger. Zusätzlich enthält TDS Laufzeitroutinen, die vom Anwendungsprogramm angeforderte E/A-Zugriffe und Zahlen-Umwandlungen übernehmen. Außerdem werden der Anschluß eines Kassetten-Rekorders für die Speicherung und das Rückrufen von Source-Programmen sowie der Anschluß eines Druckers zum Auslisten der Programme unterstützt.

Text-Editor des TDS

Der Text-Editor unterstützt den Benutzer bei der Erstellung und Bearbeitung des Quellencodes für Anwendungsprogramme. Er setzt ihn in die Lage, folgende Aufgaben auszuführen:

- Erstellung von Programmen im Quellencode für die Mikroprozessoren der Serie 32000
- Änderungen im Quellen-Text
- Darstellung von Programmzeilen aus dem Quellen-Text
- Einfügen neuer Zeilen in den Quellen-Text
- Löschen von Zeilen im Quellen-Text

Der Text-Editor erzeugt eine sequentielle Numerierung der Quellencode-Zeilen. Jedes Kommando des Text-Editors nimmt auf diese Numerierung Bezug.

TDS-Assembler

Der Assembler ist eine vereinfachte, aber kompatible Version bekannter Assembler der Serie 32000. Er akzeptiert eine spezielle Syntax für Register-List- und String-Befehle wie auch Symbole für PC-Verschiebungen, und er erzeugt einen optimierten Code für symbolisch eingegebene Verschiebungen. Die Verschiebungen werden, abhängig von ihrem Wert, als Byte, Wort oder Doppelwort codiert. Der Assembler unterstützt die symbolische Definition von Variablen und kann die einfachen Operationen ausführen, mit denen die für den Befehl „CASE" notwendige Sprungtabelle gebildet wird. Sämtliche der für Hochsprachen optimierten Adressierungsarten der Serie 32000 werden verwendet. Durch die Pseudo-Operationen FLOAT und LONG werden Gleitkomma-Operationen mit einfacher bzw. doppelter Präzision unterstützt. Der Assembler akzeptiert Zahlenwerte in dezimaler und hexadezimaler Darstellung. Die Pseudo-Operationen ermöglichen die Definition von Bytes, Worten, Doppelworten, Gleitkomma- oder langen Gleitkomma-Daten wie auch von Byte-, Wort- und Doppelwort-Blöcken. Die Pseudo-Operationen „STATIC" und „ENDSEG" unterstützen die Definition von statischen Bereichen für die Datenfelder.

Symbolischer Debugger im TDS

Nach dem fehlerhaften Assemblieren muß ein Programm initialisiert werden, bevor der Anwender es ablaufen läßt oder seinen Logikfluß mit Hilfe des symbolischen Debuggers untersuchen kann. Die Initialisierung erfolgt durch Bildung einer Modultabelle, die vier Doppelworte umfaßt. Das erste enthält die Anfangsadrese des Datenbereichs, das zweite ist die Anfangsadresse der Link-Tabelle (um zu externen Variablen oder Vorgängen zuzugreifen), das dritte ist die Anfangsadresse des Programmcodes, die letzte Eintragung ist reserviert und muß auf Null gesetzt werden.

Der wesentliche Aspekt des Debuggers ist seine Fähigkeit, mit Symbolen zu arbeiten. Er ermöglicht das Darstellen oder Ändern von Speicherzellen und der internen Register von CPU, MMU und FPU, nach Wunsch des Anwenders hexadezimal oder dezimal.

Laufzeitunterstützung

Auf alle Unterstützungsfunktionen (Run-Time-Support) des TDS-Monitors wird über Supervisor-Aufruf zugegriffen. Vorhanden sind Routinen für Ausgaben über Drucker mit Centronics-Schnittstelle, das Lesen und Zwischenspeichern von Daten oder das Schreiben gebufferter Daten für die beiden seriellen Ports wie auch Routinen für das Wandeln von binär in ASCII-Format und umgekehrt. Als Grundlage der Wandlung kann dezimales, hexadezimales, kurzes oder langes Gleitkomma-Format gewählt werden. Die Register R(0) bis R(4) der CPU werden benutzt, um die Supervisor-Aufrufe zu definieren und die Parameter zu übertragen.



Dipl.-Ing. Ulrich Schricker

Entwicklungswerkzeuge für 32-Bit-Chips

Der Schritt von 4 auf 8 und von 8 auf 16 Bit erscheint den meisten Anwendern von Mikroprozessoren heute als eine logische und notwendige Entwicklung. Die Vorteile oder gar Notwendigkeit, zu 32-Bit-Mikroprozessoren überzugehen, wird aber heute von vielen Anwendern als nicht notwendig erachtet. Der Mikroprozessormarkt wird nach wie vor von 8- und 16-Bit-Prozessoren dominiert. Doch hier ändert sich das Bild schneller als man anfänglich erwartet hat.

Der Emulator ISE 32 wird samt Cross-Software für die Serie 32000 von National Semiconductor angeboten. Er erlaubt 10-MHz-„Echtzeit"-Betrieb

Für den Schritt auf 32 Bit gibt es im wesentlichen drei Gründe:

- Die Anwender beginnen zu erkennen, daß nicht die Verdopplung der Bit-Anzahl von 16 auf 32 entscheidend ist, sondern die Architektur und die daraus resultierenden Vorteile für den Anwender das wesentliche bei dieser Entwicklung sind.
  Darüber hinaus sind wichtige Vorteile, wie z. B. Symmetrie des Befehlssatzes, Unterstützung von Hochsprachen, von Compiler-Standards, von Betriebssystemen und modularen Softwarekonzepten, die Einbindung der virtuellen Speicherverwaltung und das Slave-Prozessorkonzept heute auch mit modernen Prozessoren wie den Typen NS32016 und NS32008 für den 16- und 8-Bit-Markt verfügbar.
- Bereits heute gibt es vollständige und ausgereifte 32-Bit-Prozessorfamilien mit allen Peripheriebausteinen in Produktionsstückzahlen, z. B. die Serie NS32000.
- Die meisten Mikroprozessorhersteller beabsichtigen, 32-Bit-Typen (80386, Z80000) auf den Markt zu bringen oder haben schon mit der Einführung (MC68020) begonnen.

Dies macht es notwendig, den Markt in bezug auf Entwicklungsunterstützung für solche Prozessoren zu untersuchen. Die Hersteller von unabhängigen Entwicklungssystemen und Emulatoren haben den Eintritt der verschiedenen 32-Bit-Prozessoren auf dem Markt genau im Auge, um ihr Angebot entsprechend zu erweitern.

Der Anwender kann im allgemeinen zwischen drei Möglichkeiten wählen, wenn er sich für ein Entwicklungswerkzeug entscheiden muß: ein spezielles System, ein universelles Entwicklungssystem und Cross-Software zusammen mit einem Emulator.

Die speziellen Systeme unterstützen im allgemeinen die Prozessoren eines einzigen Halbleiterherstellers, universelle dagegen bieten Werkzeuge für die Prozessoren mehrerer Hersteller. Cross-Software wird sowohl von unabhängigen Herstellern als auch von den Halbleiterfirmen selbst angeboten. Die unabhängigen Hersteller liefern zu ihrer Cross-Software in der Regel auch Emulatoren für die verschiedenen Prozessoren.

Von National Semiconductor werden Emulatoren (ISE16, ISE32) zusammen mit der Cross-Software z. B. für die Serie 32000 angeboten. Die Verfügbarkeit solcher Emulatoren hängt im wesentlichen von der Art der Schnittstelle ab, die bestimmt, ob die Emulatoren direkt von einem Host-Rechner aus bedient und Programme von dort geladen werden können. Ist dies nicht möglich, bleibt nur die Wahl, einen speziellen Rechner zusammen mit dem Emulator als autonomes System zu benutzen.

Was muß die Entwicklungsumgebung für 32-Bit-Prozessoren leisten?

Neben den Standardfunktionen wie Softwareentwicklung und Emulation erwartet man für Prozessoren mit einer anspruchsvollen Architektur eine entsprechende Unterstützung im Entwicklungsbereich. Es macht keinen Sinn, mit Prozessoren zu arbeiten, deren Architektur auf Hochsprachenunterstützung, virtuelle Speicher-

verwaltung und modulare Programmierungstechniken zugeschnitten ist, wenn diese Eigenschaften nicht bei der Entwicklung und Emulation ebenfalls unterstützt werden.

Daraus ergeben sich folgende Forderungen:
- ein lokaler symbolischer Debugger zur Fehlersuche an Applikationssoftware im Entwicklungssystem auf Hochsprachenebene muß vorhanden sein,
- ein in „Remote"-Betrieb symbolischer Debugger zum Testen der Software innerhalb der Zielhardware ebenfalls mit Hochsprachenunterstützung sollte vorhanden sein,
- Implementierung dieses symbolischen Debuggers, ohne dabei den ausführbaren Code zu verändern oder zu verlängern, so daß Echtzeitemulation möglich ist und ebenfalls symbolisch unterstützt wird,
- Emulation der virtuellen Speicherverwaltung und virtueller Systeme,
- Unterstützung modularer Programmiertechniken, die es z. B. ermöglichen, einzelne Softwaremodule des Programmes in der Zielhardware zu ändern, ohne daß erneutes Editieren/Binden erforderlich ist,
- Betriebssystemumgebung, die besonders zur Entwicklung von Software geeignet ist (z. B. UNIX),
- eine ausreichende Anzahl von Hilfsprogrammen wie z. B. das SCCS („Source Code Control System"), das hervorragende Dokumentationseigenschaften und Schutzmechanismen für den Quellcode innerhalb eines größeren Softwareprojektes enthält,
- On-Line-Hilfsfunktionen und Handbücher, die schnelles und flexibles Handhaben auch der seltener benutzten Funktionen ermöglichen.

Der Schwerpunkt der Forderungen liegt im Bereich der Softwareunterstützung. Dies zeigt auch die zunehmende Bedeutung der Software und der Softwareeigenschaften (Architektur) der Prozessoren.

Merkmale wie Dissassemblieren, Statusanalyse, Zeitanalyse, Trace und Unterstützung von PROM-Programmierern werden heute schon von Logikanalysatoren und Emulatoren im Zusammenhang mit Entwicklungssystemen erwartet.

Cross-Software

Der Vorteil dieser Softwarepakete besteht darin, daß vorhandene Rechner genutzt und Betriebssysteme, die man schon kennt, verwendet werden können. Meist findet man die Betriebssysteme VAX/VMS, PC-DOS und CP/M, zunehmend auch UNIX. Mit einem verbreiteten Betriebssystem ergibt sich für den Anwender die Möglichkeit, Entwicklungen und Applikationssoftware auch auf andere Maschinen mit dem gleichen Betriebssystem zu übertragen.

Eine sehr weite Verbreitung im Bereich der Softwareentwicklung haben hier Universalrechner wie z. B. die

Welches Entwicklungssystem?

Die Entscheidung, welche Entwicklungswerkzeuge zur Unterstützung neuer Mikroprozessoren beschafft werden, kann von Fall zu Fall sehr unterschiedlich ausfallen. Zwei grobe Kategorien zeichnen sich ab, einmal spezielle Entwicklungswerkzeuge der Halbleiterhersteller, zum anderen Cross-Softwarepakete für universelle Rechner zusammen mit Emulatoren und Analysatoren.

Die speziellen Systeme der Halbleiterhersteller haben den großen Vorteil, daß sie einerseits immer die ersten verfügbaren Werkzeuge sind, und andererseits ein Hersteller verpflichtet ist, sein Produkt vollständig und umfassend zu unterstützen. Auch läßt sich damit die benötigte Rechnerleistung zuverlässig zur Verfügung stellen, da im allgemeinen keine anderen Applikationen zusätzlich verarbeitet werden.

Hersteller von universellen Systemen behalten sich die Entscheidung nach kommerziellen Gesichtspunkten vor, welche Prozessoren sie unterstützen. Sie geben im allgemeinen auch keine Beratung, die die Prozessoren oder Applikationsprobleme betreffen.

Dafür kann man mit diesen Systemen Entwicklungen an den Prozessoren verschiedener Hersteller unterstützen und kann Erweiterungen, wie PAL-Programmierung oder universelle Logik-Analyse, mit einbeziehen.

Die Cross-Softwarepakete bieten den Vorteil, vorhandene Rechnerkapazitäten zu nutzen. Sie unterstützen vorhandene Emulatoren (z. B. ISE32, HDS400, TEK8540).

Vorsicht ist geboten bei der Abschätzung der benötigten Rechnerkapazität für ein Entwicklungsprojekt. Meist wird die zusätzliche Belastung für universelle Rechner (z. B. VAX) unterschätzt oder die Maschine durch viele andere Applikationen bereits ausgelastet. Mit der Einführung des UNIX-Betriebssystems für Entwicklungsmaschinen schwindet der Vorteil der universellen Rechner, da unter UNIX auch „Entwicklungssysteme" zu universellen Rechnern werden.

Zur Verfügung stehen sowohl billigere Einzelplatzrechner als auch teurere Multiuser-/Timesharing-Rechner. Aufgrund der steigenden Bedeutung der Software und des wachsenden Umfangs der Softwareentwicklungen ist der Bedarf an Mehrplatzsystemen gestiegen, um gemeinsame Dateiverwaltung zu haben und Peripherie mehrfach zu nutzen.

Mit fallenden Preisen für Einplatzsysteme und Arbeitsplatzrechner wird die Bedeutung von Netzwerken in Kombination mit gemeinsamer File-Verwaltung immer größer und erscheint eher als eine Notwendigkeit in Anbetracht der steigenden Bedeutung der Software. Das gilt insbesondere auch für Projektverwaltungswerkzeuge.

Qualität und der Umfang der Softwarewerkzeuge sind inzwischen ein zentrales Bewertungskriterium für die Entscheidung zu einem Entwicklungssystem und auch für oder gegen eine Prozessorfamilie geworden; denn nicht mehr Taktfrequenz oder „MIPS" entscheiden über die Leistungsfähigkeit eines Prozessors in einer Systemumgebung, sondern die Prozessorarchitektur in Verbindung mit dem Systemkonzept.

VAX gefunden. Viele Anwender nutzen diese Maschine auch für Chip-Entwurf, Simulation, Platinenentwurf und CAE-Anwendungen. So ist es leichter, die hohen Investitionskosten zu rechtfertigen.

Aber es zeigt sich, daß z. B. eine VAX750 mit sechs bis acht Benutzern, die im Editor und Compiler arbeiten, ausgelastet ist. Man möchte schließlich noch mit akzeptablen Reaktionszeiten arbeiten. Die Universalität dieser Rechner hat häufig zur Folge, daß die Rechner bereits ausgelastet sind und es keinen Sinn macht, zusätzlich ein größeres Softwareprojekt auf der Maschine bearbeiten zu wollen. Wohl können solche Rechner als zentraler Netzwerkknoten oder als „Fileserver" zusammen mit mehreren Entwicklungsstationen arbeiten. Entsprechende Kommunikationsverbindungen werden inzwischen von den meisten unabhängigen Produzenten und den Halbleiterherstellern zwischen ihren Systemen und der VAX angeboten.

Die meisten Cross-Software-Pakete für 32-Bit-Rechner, die auch die vollständige Unterstützung der Emulatoren und die Kommunikation mit ihnen enthalten, werden heute für die Prozessorfamilie Serie 32000 angeboten. Es stehen Softwarepakete von verschiedenen Herstellern zur Verfügung, die unter CP/M, PC-DOS, RSX-11 (PDP-11), VMS (VAX) und UNIX (VAX) laufen.

Universelle Systeme

Universelle Systeme sind für Logikanalyse, Emulation und Softwareentwicklung vorgesehen. Sie unterstützen heute die meisten 8- und 16-Bit-Prozessoren und werden auch die 32-Bit-Chips unterstützen.

Das Entwicklungssystem HP64000 (Hewlett-Packard) arbeitet als autonomes System zur Softwareentwicklung und kann durch zusätzliche Platinen um Status-, Zeitanalyse, Emulation und Emulationsspeicher erweitert werden. Es ist auch geplant, die Softwarewerkzeuge für die VAX zur Verfügung zu stellen und ein Interface zur VAX einzuführen, das das Einbinden der HP64000-Stationen in ein Netzwerk mit der VAX ermöglicht.

Im allgemeinen schreibt der Benutzer sein Programm in Pascal, und für die meisten Prozessoren steht ein Assembler zur Verfügung. Zusätzlich wird ein Softwareanalysator angeboten, der es erleichtert, Assemblerprogramme zu optimieren.

Die Entwicklungssysteme der TEK8500-Serie (Tektronix) arbeiten zur Softwareentwicklung mit einer VAX zusammen, können aber auch eigenständig arbeiten. Die Serie besteht aus mehreren Einheiten, die zum Test und Debugging der Software geeignet sind oder als Emulator arbeiten und untereinander kommunizieren können. In dieses Konzept ist auch das „Digital Analysis System" DAS 9100 eingebunden.

Mit den Systemen ATLAS und COLT (Dolch Logic Instruments) läßt sich die Software ebenfalls auf einer VAX oder PDP-11 entwickeln und anschließend in die Systeme laden. Aber auch Stand-Alone-Betrieb ist möglich. Die Systeme können für die jeweilige Anwendung durch verschiedene Einschübe konfiguriert werden, einschließlich Logikanalyse, Emulation, EPROM- und PAL-Programmierung, Wortgeneratoren, Signatur- und Signalformanalyse.

Mit den Entwicklungssystemen KDS (Kontron) werden Logikanalyse, Softwareentwicklung, PROM-Programmierer und Emulatoren unterstützt. Es stehen mehrere Ausbaustufen zur Verfügung bis zu einem System unter UNIX, an dem drei Benutzer gleichzeitig arbeiten können. Pascal- und C-Compiler stehen ebenso zur Verfügung, wie die Möglichkeit, mehrere Stationen über Ethernet zu verbinden.

Für die Emulatoren der Serie KSE existiert auch ein Interface zu VAX- und PDP-11-Rechnern, über das Software, die auf diesen Rechnern entwickelt wurde, direkt geladen werden kann. Mit einer zusätzlichen Hardware (KDA) lassen sich die Emulatoren auch über einen PC steuern.

Einen gravierenden Einfluß auf die Verfügbarkeit von Emulatoren für die universellen Systeme hat die Unterstützung durch die Halbleiterhersteller. So ist z. B. der Prozessor 80286 auf dem Markt nur in einer Version erhältlich, die den Zugriff auf interne Register und Steuersignale nicht ermöglicht. Aber ohne diesen Zugriff ist die Konstruktion eines Emulators extrem schwierig, wenn nicht gar unmöglich. Natürlich hat der Hersteller dieses Prozessors spezielle Versionen des Chips, bei dem alle Signale zugänglich sind, um eigene Emulatoren zu bauen und zu verkaufen.

Spezielle Systeme

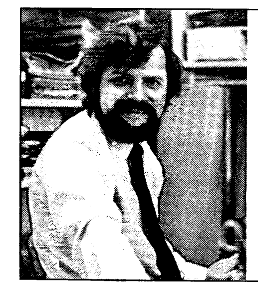
Halbleiterhersteller sind gezwungen, ihre neuen Prozessoren möglichst frühzeitig zu unterstützen und eine vollständige Entwicklungsumgebung anzubieten. Daher sind die speziellen Systeme immer als erste Entwicklungssysteme einschließlich der Emulatoren auf dem Markt. Das stellt einen ganz gravierenden Vorteil für den Anwender dar.

Prozessoren, Slave-Prozessoren und periphere Bausteine der Serie 32000 stellen die bis jetzt einzige vollständige 32-Bit-Familie auf dem Markt dar. Für sie wird eine umfassende Entwicklungsunterstützung angeboten, die auf dem kompletten 32000-Chipsatz aufgebaut ist und in allen Punkten den Forderungen an eine 32-Bit-Entwicklungsumgebung entspricht. Anhand dieses Beispiels sollen 32-Bit-Entwicklungswerkzeuge näher vorgestellt werden.

32000-Entwicklungsunterstützung

Es steht ein Multiuser/Multitasking-Entwicklungssystem (SYS32) zur Verfügung, das acht Benutzer gleichzeitig bedient und als erster auf einem Mikroprozessor basierender Rechner die volle virtuelle Speicherverwaltung mit Demand-Paged-Zugriff implementiert hat und in seiner Leistung einer VAX750 entspricht.

Es wird mit GENIX4.1, das auf Berkley-UNIX4.1 zurückgeht, geliefert, enthält 1,25 MByte RAM mit Fehlerkorrektur, einen „intelligenten" Schnittstellenmulti-



Dipl.-Ing. Ulrich Schricker ist in Augsburg geboren und studierte an der TH Aachen. Dort hat er anschließend zwei Forschungsaufgaben aus dem Bereich der Automatisierung und Lasermeßtechnik am Institut für Strahlantriebe bearbeitet. Dann ging er zur Firma Polytec und arbeitete an der Entwicklung von Multiprozessorsystemen. Anfang 1983 wechselte er zu National Semiconductor und betreute das zentrale Applikationslabor. Er ist seit Ende 1984 für das europäische Marketing der Entwicklungswerkzeuge für 8-/16-/32-Bit-Prozessoren verantwortlich.
Hobby: Kochen

plexer für acht Terminals, 70 MByte Hard-Disk und eine 20-MByte-Streamer-Bandmaschine in der Minimalkonfiguration.

GENIX bietet über 200 Hilfsprogramme, wie SCCS („Source Code Control System") und Projektverwaltungswerkzeuge, Textverarbeitungs- und Dokumentationsprogramme und einen C-Compiler.

Die Entwicklungssoftware enthält einen lokalen und einen symbolischen „Remote"-Debugger, der voll die Emulatoren (ISE16, ISE32) unterstützt und als einziger das symbolische Debugging von ausführbarem Code erlaubt, ohne den Code zu ändern oder mit Debugg-Informationen zu verlängern. Somit ist es möglich, auch Echtzeitemulation mit voller symbolischer Unterstützung durchzuführen.

Das Einplatzentwicklungssystem VR32 bietet die gleiche Unterstützung an. Dieses System verwendet ebenfalls den kompletten 32000-Chipsatz und enthält 1 MByte RAM, 1 MByte Floppy- und 40 MByte Harddisk-Kapazität sowie das Betriebssystem Unix in der Version 2.0.

Es stehen Emulatoren für NS32016 und NS32032 zur Verfügung. Hier soll als Beispiel auf den Emulator ISE32 für den NS32032 eingegangen werden, dessen Eigenschaften im wesentlichen denen des Emulators ISE16 entsprechen.

Zusammen mit einem Hostsystem VAX, SYS32 oder VR32 emuliert der „In System Emulator" ISE 32 den vollen Chipsatz einschließlich CPU NS32032, virtuellem Speicherverwaltungsbaustein NS32082 (MMU) und Zeitsteuereinheit NS32201 (TCU).

Die MMU (Memory Management Unit) läßt sich zur Emulation zu- oder abschalten. Der ISE32 ermöglicht es dem Anwender, sowohl Hardware als auch Software im eigenen Zielsystem, das auf der Serie 32000 aufgebaut ist, zu testen und fehlerfrei zu machen. Er kann auch seine Software nur in dem Emulator testen und debuggen, wenn seine Hardware noch nicht fertig ist. Der Emulator arbeitet entweder in der Emulatorbetriebsart, während ein Benutzerprogramm emuliert wird, oder im „Monitor"-Betrieb, wenn der Benutzer über den Hostrechner mit dem Emulator kommuniziert.

Der Emulator ISE32 enthält einen internen Taktgenerator für unterschiedliche Frequenzen bis zu 10 MHz und einen schnellen 128 KByte großen Speicher, der Echtzeitemulation ermöglicht. Es sind alle Möglichkeiten vorhanden, um die Emulation anzuhalten, CPU-Register, Slaveprozessor-Register und Speicherbereiche zu testen und zu ändern.

Der Emulator ISE32 besteht aus einer Hardware mit RS-232-Schnittstellen für den Rechner und ein Terminal, einem Monitor-Firmware-Programm, das sich in PROMs im Emulator befindet und einem symbolischen ISE-Debugger (IDBG32), der Teil der Entwicklungssoftware auf dem Hostrechner ist.

Der ISE32 hat vier Hardware-Unterbrechungspunkte mit 76 Bit. Es sind alle Adressen, Daten, Status-Bits und/oder acht externe TTL-Signale (über den TTL-Status-POD) enthalten, die zur Bildung der Unterbrechungsbedingung herangezogen werden können. Unterbrechungspunkte lassen sich entweder kombinatorisch oder sequentiell verbinden (z. B. A oder B, B vor C usw). Zusätzlich können noch Unterbrechungspunkte über einen ganzen Speicherbereich erzeugt werden („breakrange"), mit dem ein Benutzer einen Stopp auslösen kann, wenn eine Adresse aus einem definierten Speicherbereich auftaucht.

Der ISE32 hat 32-Bit-Zähler und -Zeitgeber, um Zeitmessungen zwischen zwei Ereignissen vornehmen zu können und/oder um nach einer vorgegebenen Zahl von Ereignissen ein neues Ereignis zu erzeugen. Auf diese Weise kann man nicht nur Zeitmessungen mit Taktzyklen, Ereignissen, Befehlsausführung oder Speicherzyklen machen, sondern den Zeitgeber/Zähler durch ein Ereignis einschalten und durch ein anderes ausschalten. Damit lassen sich auch Statistiken zur Programmoptimierung erstellen.

Der Emulator hat 1023 × 128 Trace-Speicher, um den Programmfluß („non-sequential"-Befehle oder -Verzweigungen) oder Bus-Status-Analyse aufzuzeichnen. Der Eintrag in den Speicher enthält Adressen, Daten, Status, externe TTL-Signale und einiges mehr.

Da die Architektur des NS32032 Multiprozessoranwendungen vorsieht, wird auch diese Eigenschaft von dem Emulator unterstützt. So kann der ISE32 nicht nur zur Emulation von asynchronen Mehrrechnersystemen, die gemeinsame und private Bereiche verwenden, benutzt werden, sondern auch zur Emulation von zwei synchron gekoppelten Rechnern, was eine besondere Eigenschaft der CPU NS32032 darstellt.

Ulrich Schricker

Leistungsfähiges Entwicklungswerkzeug für die 32000-Familie:

Zweiplatzentwicklungs- und Prototypensystem VR32

Bei der Entscheidung über die Verwendung eines neuen Mikroprozessors spielt neben der Leistungsfähigkeit des Prozessors selbst die Verfügbarkeit einer entsprechenden Entwicklungsunterstützung eine wesentliche Rolle. Der Faktor Entwicklungsumgebung bestimmt erheblich die Kosten für den Ein- oder Umstieg auf einen neuen Mikroprozessor-Typ. National Semiconductor unterstützt die Anwender von Bausteinen der 32000-Familie durch eine Reihe von Möglichkeiten: neben kompletten kleinen (VR32-) und großen (SYS32-)Entwicklungssystemen sind Cross-Software-Pakete für IBM-PC und VAX unter VMS oder Unix lieferbar. Software-Häuser bieten unter Lizenz diese Entwicklungs-Software auch für Z80/CP/M- und PDP11/RSX11-Systeme an. Mit Zusatz-Hardware läßt sich auch unter Isis-II auf Intel-Entwicklungssystemen Hard- und Software für die Serie 32000 entwickeln.

Bedarfsgerechte Entwicklungsunterstützung

Die verschiedenen Entwicklungspakete haben unterschiedliche Leistungsmerkmale, wobei die Leistungsfähigkeit stark vom jeweils verwendeten Hostrechner abhängt. Natürlich variieren auch die Kosten entsprechend. Im mittleren Preisniveau bietet National Semiconductor eine leistungsfähige und professionelle Entwicklungsstation mit zwei Arbeitsplätzen an: das System VR32.

VR32 ist ein modulares Zweiplatz-Entwicklungssystem für den kostengünstigen Einstieg. Wegen seiner Multibus-I-Kompatibilität ist es auch als Prototyp-System einsetzbar. Es bietet zusammen mit weiteren Entwicklungswerkzeugen für Software, Hardware und Emulation eine vollständige Entwicklungsumgebung, die bei wachsendem Bedarf ausgebaut werden kann. Die Module des Systems VR32 lassen sich einfach aufeinanderstapeln. Das Basismodul beinhaltet einen Multibus-Rahmen für drei Standardplatinen des Computers und drei freie Steckplätze. Das zweite Modul des Standard-Lieferumfangs ist das Diskmodul mit einer 40-MByte-Winchesterplatte und einem 1-MByte-Floppy-Disk-Laufwerk. Das dritte Modul ist optionell lieferbar und beinhaltet ein 20-MByte-Magnetband-Laufwerk für ¼-Zoll-Kassetten. Das Streamer-Format ist mit den anderen Laufwerken der National Semiconductor Entwicklungssysteme kompatibel.



Bild 1. Das Entwicklungssystem SYS 32 unterstützt die Verwendung von Hochsprachen und modulare Programmierung

VR32 wird mit dem Multiuser/Multitasking-Betriebssystem Unix-V/Serie 32000 geliefert. Darüber hinaus sind im Lieferumfang alle für eine professionelle Entwicklungsumgebung erforderlichen Software-Werkzeuge und -Hilfsmittel enthalten. Außer den zusätzlichen Compilern gibt es keine Optionen, die zusätzlich bestellt oder bezahlt werden müßten. Das System wird komplett geliefert, um komfortable Programme zu entwerfen, zu testen, zu dokumentieren und zu archivieren. Standardmäßig sind desweiteren symbolische Debugger und die Steuerungs-Software für die In-System-Emulatoren im Lieferumfang enthalten.

Zum Thema Debugger

Debugger sind als Software-Werkzeuge von großer Bedeutung, wenn es um die Anwenderfreundlichkeit eines Entwicklungssystems geht. Um den Begriff „symbolischer Debugger" gibt es nicht selten ein Verwirrspiel. Ist ein symbolischer Debugger das gleiche wie ein „Source-Level-Debugger"? Oder, was ist überhaupt ein „Software-Debugger"?

Als Erklärungsbeispiel soll ein Debugger dienen, mit dem eine Stelle im Speicher betrachtet wird. Welche Informationen stellt dieser Debugger zur Verfügung?

Ein Source-Level-Debugger kann die jeweilige Stelle im Quellcode finden und die Zeile aus dem Quellcode (z. B. in C-, Pascal- oder Assembler-Sprache) darstellen, aus der der Compiler oder Assembler den Maschinencode generiert hat, der in der betreffenden Speicherstelle steht. Der Debugger ist hierzu nur in der Lage, wenn sowohl Compiler, Assembler, Linker als auch das Objekt-File-Format darauf ausgelegt sind, diese Eigenschaften zu unterstützen und entsprechend zusammenzuarbeiten. Compiler und Assembler müssen in der Objektdatei Zeiger ablegen, die es dem Debugger ermöglichen, die zugehörigen Quellcodedateien zu finden und die betreffende Zeile wiederzugeben.

Ist ein symbolischer Debugger im Einsatz, kann er Informationen über die Symbole aus dem Quellcode finden und darstellen. Man muß jedoch unterscheiden, auf welche Symbole der Debugger zugreifen kann: auf lokale oder auf externe Symbole? Wenn keine Referenzen zu Objekten oder Stellen außerhalb der Datei bestehen und weder der Linker noch eine andere Quellcodedatei das Symbol benutzen, so ist es ein lokales Symbol. Das sind beispielsweise Namen von temporären Variablen, die nur in der speziellen Datei benutzt werden, oder die Labels von GOTO-Anweisungen. Ein externes (oder globales) Symbol wird auch durch andere Quellcodedateien und den Linker genutzt. Informationen über externe Symbole werden vom Compiler oder Assembler in den Objektdateien abgelegt, damit der Linker die Referenzen zwischen den einzelnen Quellcodedateien während des Verbindens richtig bearbeiten kann. Beispiele für externe Symbole sind Namen von Prozeduren, die durch andere Prozeduren aufgerufen werden können, oder Variable, die in einer Quellcodedatei definiert sind, jedoch auch in anderen Dateien benutzt werden. Die Eigenschaft, ob ein Symbol lokal oder extern ist, wird selbstverständlich vom Programmierer definiert.

Für die symbolischen Debugger bedeutet dies, daß sie immer externe Symbole finden und darstellen können, da die dazu notwendige Information vom Assembler oder Compiler in der Objektdatei abgelegt ist. Über die lokalen Symbole kann der Debugger jedoch nur Informationen finden und darstellen, wenn das Format der Objektdateien („Object File Format") so definiert wurde, daß es auch darüber Informationen aufnehmen kann.

Weil Quellcode-Debugger den vollständigen Zugriff auf den Quellcode haben, können sie sowohl externe als auch lokale Symbole finden und darstellen.

Der Software-Debugger ist im Prinzip ein Marketing-Produkt. Jeder Debugger tut das, wozu er geschrieben wurde: er bearbeitet Software. Hat der Debugger einen Disassembler, so erzeugt er einen Assemblercode, aus dem beim Assemblieren der untersuchte Maschinencode wieder entstehen würde. Der durch den Disassembler erzeugte Code kann deutlich vom ursprünglichen Quellcode abweichen, auch wenn dieser in Assembler geschrieben wurde. Das geschieht deshalb, weil der Disassembler nur die „nackten" Bits und Bytes des Maschinencodes nutzt, jedoch nicht die Symbole, die der Programmierer definiert und verwendet hat, um seine Programme zu organisieren und zu schreiben.

Ein standardisiertes Objektformat

Das Format der Objektdateien (Object File Format) ist, das geht aus den vorherigen Ausführungen hervor, der wesentliche Faktor für die Qualität der Software-Entwicklungsumgebung. National Semiconductor hat sich zusammen mit AT&T und anderen Computer-Herstellern bemüht, ein standardisiertes Format für die Objektdateien zu schaffen: das COFF (Common Object File Format). Alle Software-Werkzeuge, wie z. B. Compiler, Assembler und Linker des VR32-Systems arbeiten auf der Basis von COFF.

DBG16-, IDBG16- und der IDBG32-Debugger sind Quellcode-Debugger, die es erlauben, Software in einer externen Hardware (Zielsystem) zu „debuggen". Das besondere daran ist, daß trotz der komfortablen Möglichkeiten keine Veränderungen oder die Verlängerung des ausführbaren Codes durch die zusätzlichen Informationen für den Debugger stattfinden. Alle erforderlichen Informationen, um symbolisches Debugging auf der Quellcode-Ebene zu realisieren, sind im Hostrechner in gesonderten Referenztabellen abgelegt. DBG und IDBG ermöglichen es, gleichzeitig Module, die in verschiedenen Sprachen (Assembler, Pascal, C, F77) geschrieben wurden, komfortabel auf Quellcode-Ebene zu debuggen.

Echtzeitbetrieb und „Source-Level-Debugging"

Die umsichtige Definition des Formates für die Objektdateien bewirkt, daß auch zusammen mit den

Emulatoren ISE16 und ISE32 in Echtzeit emuliert werden kann und gleichzeitig die volle, komfortable Unterstützung des Quellcode-Debuggers zur Verfügung steht. Für eine Betrachtung ist eine Echtzeit-Anwendung interessant, die auf dem EXEC-Echtzeitkern von National Semiconductor beruht. EXEC hat Schnittstellen zu den Hochsprachen Pascal sowie C, und der Anwender wird es vorziehen, seine Applikationen in einer dieser Sprachen zu schreiben. Da die Prozessoren der Serie 32000 durch ihren Befehlssatz Hochsprachen-Unterstützung bieten, ist eine Verbesserung des Zeitverhaltens auf Assembler-Ebene nur in geringem Maße nötig. Der Anwender kann somit in seiner Zielhardware die CPU und TCU mit einem der Emulatoren ISE16 oder ISE32 emulieren, so daß sein Programm ohne Einschränkungen in Echtzeit abläuft. Dabei ist es möglich, die Namen der Variablen, Labels oder Zeilennummern aus dem Quellprogramm, das z. B. in C geschrieben wurde, zum Debuggen zu benutzen. Der ISE16-Emulator bildet die NS32016-CPU, die MMU und die TCU nach, der ISE32-Emulator die NS32032-CPU, die MMU und die TCU.

Das „Source-Code-Control-System"

Das Source-Code-Control-System (SCCS) ist ein Werkzeug, um die Entwicklung der Quellcode-Module zu überwachen und zu dokumentieren. Es werden nur die Veränderungen von einer Version zur nächsten abgespeichert, aus denen bei Bedarf jede beliebige ältere Version wieder rekonstruiert werden kann. Für das SCCS stehen rund 20 separate Befehle zur Verfügung.

VR32-Hardware

Herzstück der Hardware des VR32-Systems ist die CPU-Platine mit dem kompletten 10-MHz-Chipsatz der Serie 32000 (NS32016-CPU). Darin enthalten sind die MMU (Memory Management Unit) zur virtuellen Speicherverwaltung nach dem „Demand-Paged"-Verfahren, die FPU (Floating Point Unit) für schnelle Fließkomma-Operationen, die ICU (Interrupt Control Unit) zur Steuerung und Kontrolle der Interrupts und die TCU (Timing Control Unit).

Die NS32016-CPU ist genau wie die anderen CPU-Typen der Serie 32000 eine komplette 32-Bit-Zentraleinheit. Sie unterscheidet sich von den anderen Zentraleinheiten lediglich dadurch, daß sie zum externen Anschluß einen 16 Bit breiten Datenbus hat. Alle CPU-Typen der Familie 32000 sind, nebenbei bemerkt, vollständig softwareauf- und -abwärtskompatibel.

Auf der CPU-Platine gibt es darüber hinaus zwei serielle und eine parallele Schnittstelle sowie einen 0,5-MByte-Arbeitsspeicher. Centronics-kompatible Drucker lassen sich an eine entsprechende Parallel-Schnittstelle anschließen. Auf einer anderen Platine des CPU-Moduls sind weitere 0,5 MByte RAM untergebracht, was einen



Bild 2. VR32 ist ein vollständiges Entwicklungssystem für die Prozessoren der 32000-Familie mit Emulator

Arbeitsspeicher von insgesamt 1 MByte in der Standard-Konfiguration ergibt.

Die Disk-/Band-Steuerplatine kann zwei Festplatten-, zwei Floppy-Disk- und ein Streamer-Laufwerk steuern. Der Anschluß erfolgt über die Standard-Schnittstellen ST506 für die Festplatten, SA460 für die Floppy-Disks und QIC-02 für das Bandlaufwerk.

Im Diskmodul befinden sich eine 40-MByte-Festplatte und ein 1-MByte-Floppy-Disk-Laufwerk, jeweils in 5,25-Zoll-Technik. Das Tapemodul ist mit einem 20-MByte-Streamerbandlaufwerk für ¼-Zoll-Kassetten ausgerüstet. Alle Module sind mit Pin-zu-Pin-Flachkabeln verbunden und mechanisch arretiert, wenn sie aufeinandergestapelt werden.

Das VR32-System hat zwei RS232C-Schnittstellen zum Anschluß von Terminals, In-System-Emulatoren (ISE16 oder ISE32), seriellen Druckern oder anderen seriellen E/A-Geräten.

Betriebssystem VR32

Das VR32 wird mit einem „Unix-System-V/Serie-32"-Multiuser/-Multitasking-Timesharing-Betriebssystem ausgeliefert. Die Implementation von System V für die Serie 32000 beinhaltet wesentliche Verbesserungen:

- Paging – Die virtuelle Speicherverwaltung nach dem „Demand-Paged"-Verfahren ersetzt diejenige nach dem segmentierten Swap-Mechanismus. Das Paging-Verfahren lädt seitenweise (1 KByte) nur die Programmteile, die augenblicklich von den laufenden Prozessen benötigt werden. Unbenutzte Seiten werden automatisch in den Massenspeicher zurückgeladen, und die freien Seiten im Arbeitsspeicher können mit anderen Programmteilen belegt werden.
- Record- und File-Locking – eine Synchronisationsmethode, die es in einem Prozeß möglich macht, entweder einen Datenbereich für gemeinsames Lesen durch

mehrere Benutzer zu reservieren (read locked, share) oder ihn zum Beschreiben für einen Benutzer freizugeben und für die anderen zu blockieren (write locked, exclusive). Diese Eigenschaft ist für den Aufbau von Datenbanken wichtig, um die Zugriffe auf die Datenbereiche durch verschiedene Benutzer zu regeln.
- Erweiterungen des Software-Generation-Systems – diese Erweiterungen betreffen vor allem das Zusammenspiel zwischen den Compilern und den Bibliotheken im COFF (Common-Object-File-)Format.
- Neue Kommandos – zum Beispiel „trenter", um Fehlerbeschreibungen direkt an das Unix-System-Support-Center schicken zu können. Viele Befehle sind um zusätzliche Optionen erweitert.
- Curses/Terminfo-Package – ein Paket von Unterprogrammen zur Verwaltung von Bildschirmen. Terminfo ist ein System von Dateien, in denen die Hardware-Eigenschaften und die Escape-Sequenzen der verschiedenen Terminaltypen beschrieben sind. Das Curses/Terminfo-Paket erlaubt dem Programmierer, terminalunabhängige Software zu schreiben, die in der Lage ist, die terminalabhängigen Eigenschaften, die in Terminfo beschrieben sind, zu nutzen.

Entwicklungssoftware

Die Entwicklungssoftware des VR32 beinhaltet:
- F77-Standard-System-V/Serie-32000-Fortran-Compiler, der die Ansi-Fortran-77-Forderungen erfüllt.
- C-Compiler (optimiert);
- Pascal-Compiler, kompatibel zum ISO-Standard (optional).
- Serie-32000-Assembler;
- DBG16-COFF-Debugger. Ein symbolischer Source-Level-Debugger, der vollständig auf der Ebene des Quellcodes arbeitet. Er ist sowohl lokal als auch „remote" zusammen mit einer Zielhardware (z. B. NS-Entwicklungsplatine DB32016 oder DB32000) ablauffähig. Mit ihm ist es möglich, gleichzeitig verschiedene Module, die in unterschiedlichen Sprachen geschrieben sind, auf Hochsprachenebene zu debuggen.
- IDBG16 und IDBG32 – symbolische Source-Level-Debugger für die In-System-Emulatoren ISE16 und ISE32. Hiermit wird auch die Funktion der Emulatoren gesteuert. Programme können in die Emulatoren geladen und dort ablaufen oder über die Emulatoren in die Zielhardware geladen, dort ausgeführt, aufgezeichnet, verfolgt und getestet werden. Der IDBG hat die gleiche Benutzeroberfläche und die gleichen Hochspracheneigenschaften wie der DBG16-Debugger. Testsoftware für den ISE16- und den ISE32-Emulator sind ebenfalls erhältlich.
- DDT-System-Debugger – zur Auswertung der Dump-Bereiche, die vom System bei Absturz eines Programms angelegt werden.
- Runtime-Support-Bibliothek;
- Linker;
- „nburn" – Hilfsprogramm zum Anschluß von PROM- und EPROM-Programmiergeräten.
- MON16- und MON32-Monitore – in der Zielhardware erforderlich, um mit DBG16 zu kommunizieren. Die Monitore enthalten ein Terminal-Modul, das den transparenten Betrieb mit dem Hostrechner unterstützt, eine Laufzeitumgebung, um Programme, die vom Host in die Zielhardware geladen wurden, auszuführen und ein Debugger-Modul, das zusammen mit dem DBG16-Debugger des Hostrechners arbeitet. Die Monitore lassen sich für beliebige Zielhardware konfigurieren.

Günter Hausmann

ADA-Entwicklungssystem für die 32000-Familie

Obwohl die Hersteller von Workstations und kommerziellen Computern in größerem Maßstab die ersten Anwender der 32000-Serie waren, bietet die Architektur dieser Prozessor-Familie auch in anderen Applikationsbereichen wie Robotertechnik, Verteidigungs- und anderen Hochleistungs- oder Echtzeit-Systemen große Vorteile. Zur Unterstützung solcher Applikationen bietet National Semiconductor eine Reihe von Hardware-Optionen für militärische Zwecke sowie – was noch wichtiger ist – das „Verdix-ADA-Entwicklungssystem" (VADS) an, das in diesem Beitrag näher beleuchtet wird.

Konzipiert vom amerikanischen Verteidigungsministerium (Department of Defense, DOD) wird ADA weitgehend als militärisch orientierte Sprache betrachtet und in einer wachsenden Zahl von militärischen Bereichen eingesetzt. Dabei leiten sich die Vorteile von ADA in diesen Bereichen nicht von irgendwelchen militärspezifischen Merkmalen dieser Sprache ab, sondern von der Art, wie ADA die Entwicklung von qualitativ guter, zuverlässiger Software unterstützt. Der Zweck dieses Beitrages ist deshalb ein dreifacher: einen kurzen historischen Abriß von ADA zu geben, die Merkmale näher zu untersuchen, die ADA so wichtig machen und die Möglichkeiten zu beschreiben, die das VADS-System bietet.

Die Software-Krise

Die Entwicklung von ADA wurde durch das amerikanische Verteidigiungsministerium als Antwort auf die sogenannte „Software-Krise" initiiert. Den rapide sinkenden realen Kosten für Hardware standen keinerlei Kostenreduzierungen bei der entsprechenden Software gegenüber. Im Gegenteil, die Software-Kosten eskalierten mit der Komplexität und der wachsenden Zahl der Projekte. Die Software-Krise ist ein Bündel von Problemen, das die zunehmende Unfähigkeit der Entwickler widerspiegelt, den heutigen Erfordernissen gerecht zu werden. Diese Probleme betreffen nicht nur die Kosten von Software, sondern auch deren Qualität und Zuverlässigkeit.

So erfüllt beispielsweise ein unakzeptabel großer Teil von Software nicht die Spezifikationen, wird zu spät fertig oder beides zusammen. Wo Software durch den Anwender geschrieben wird, sind Kosten und Zeitplan schwierig zu steuern und werden oft überschritten. Existierender Code, der wiederverwertet werden könnte, wird oft nicht benutzt, und es ist häufig unmöglich, Programme auf verschiedene Zielrechner zu portieren.

Auch wenn eine Software komplett ist und offensichtlich erfolgreich arbeitet, sind die Wartungskosten hoch. Die meiste Software beinhaltet Fehler, die nicht unmittelbar entdeckt werden und es ist nicht ungewöhnlich, daß die ursprünglichen Spezifikationen geändert werden, nachdem man praktische Erfahrungen mit dem Produkt gemacht hat. Fehlerbehebung und andere Modifikationen sind teuer und können darüber hinaus neue Fehler hervorbringen. Typischerweise können die Wartungskosten sich auf mehr als die Hälfte der gesamten Software-Kosten summieren.

Das amerikanische Verteidigungsministerium fand zwei prinzipielle Ursachen für die Software-Krise heraus. Eine Ursache besteht darin, daß in militärischen Systemen mehr als 500 nichtkompatible Sprachen im Einsatz waren. Die andere Ursache war, daß keine dieser Sprachen speziell darauf ausgelegt war, eine Software-Entwicklungstechnik zu unterstützen, die dabei hilft, die Qualität und Zuverlässigkeit von Software zu verbessern. Darüber hinaus konnte keine der vorhandenen Sprachen den Erfordernissen gerecht werden, die aus der Vielzahl unterschiedlicher Applikationen entstanden.

Das DOD entschloß sich deshalb, die Entwicklung einer vollständig neuen Sprache zu fördern, die in effizienter Weise im gesamten Spektrum der vorhandenen Computer-Applikationen einsetzbar wäre. Die Verringerung der im Gebrauch befindlichen Sprachen auf eine einzige würde natürlich sofort Gewinne bringen. Aber es galt auch zu entscheiden, ob die Gelegenheit ergriffen würde, in die neue Sprache Möglichkeiten einzubauen, die alle anderen Probleme eliminieren oder reduzieren, die mit der Entwicklung von großen, komplexen Programmen verbunden sind.

Wie *Tabelle 1* zeigt, waren die Entwicklungsziele sehr anspruchsvoll, und es erforderte verschiedene Untersuchungen, um herauszufinden, ob es möglich ist, eine Sprache zu entwickeln, die den Anforderungen gerecht wird. 1977 wurden Anbieter eingeladen, um den Entwicklungsvertrag abzuschließen, der dann letztendlich an die Firma CII Honeywell Bull ging. 1980 war das DOD hinreichend von der „Lebensfähigkeit" der Sprache ADA überzeugt und machte sie verbindlich für alle projektbezogenen Systeme. 1983 wurde ADA mit Wirkung vom Januar 1986 an in der NATO eingeführt.

Tabelle: Entwicklungsziele der Sprache ADA

- Verbesserung der Software-Qualität und -Zuverlässigkeit;
- Verminderung der Wartungskosten;
- Portierungsfähigkeit;
- Unterstützung für die strukturierte Programm-Entwicklung;
- Eignung für komplexe, eingebettete Systeme;
- Code-Wiederverwendbarkeit;
- Unterstützung für große Entwicklungs-Teams.

ADA ist mehr als nur eine weitere Programmiersprache. Es ist ein genereller Ansatz zur Lösung der Probleme bei der Erstellung komplexer Programme, die zuverlässig arbeiten sollen. Weil die Zuverlässigkeit eine so wichtige Größe ist, hat das DOD die „ADA-Compiler-Validation-Capability (ACVC)" geschaffen, ein Testprogramm mit mehr als 2000 Sprachtests, das ständig aktualisiert und erweitert wird. Nur Compiler, die diese strengen Tests überstanden haben, können als ADA-Compiler verkauft werden.

Obwohl ADA-Compiler in der Regel zuerst auf VAX-Computern implementiert werden, liegt der Hauptmarkt langfristig bei Mikrocomputer-Zielmaschinen. Zur Zeit ist der Einsatz von ADA im wesentlichen auf militärische Applikationen beschränkt, wo die Verwendung verbindlich ist. Es besteht allgemein Übereinstimmung darin, daß der Markt innerhalb von zehn Jahren auf etwa 10 Mrd. $ wachsen wird. Der Einsatz von ADA im kommerziellen Bereich wird natürlich langsamer vorankommen, aber man erwartet eine Marktgröße von mehr als 100 Mrd. $ Mitte der 90er Jahre.

ADA im Überblick

Während die Zukunft von ADA bei militärischen Applikationen gesichert ist, muß die Sprache im kommerziellen Bereich mit Pascal, „C" und anderen verbreiteten Sprachen konkurrieren, und sie wird nur Erfolg haben, wenn sie offensichtliche kommerzielle Vorteile bietet. Die meisten Programmiersprachen wurden unter der Prämisse entwickelt, einfach erlernbar oder einfach implementierbar zu sein. Im Gegensatz dazu standen bei der Entwicklung von ADA Leistungsfähigkeit und Zuverlässigkeit im Vordergrund. Daraus folgt, daß ADA nicht leicht zu implementieren ist und einen höheren Lernaufwand erfordert. Erste Ergebnisse bei militärischen Applikationen zeigen jedoch erhebliche Verbesserungen in der Produktivität. Ein Vertragspartner zum Beispiel war in der Lage, seine Preise um 25 % zu reduzieren und eine unbegrenzte Garantie für seine Software zu bieten.

ADA ist in hohem Maße aus denselben Gründen für kommerzielle Applikationen ebenso geeignet wie für militärische. Die Sprache unterstützt die strukturierte, modulare Programmierung und wiederverwendbare Software-Komponenten. Sie vereinfacht die Probleme, die mit der Abwicklung großer Projekte verbunden sind und steigert sowohl die Programmierer-Produktivität als auch die Software-Qualität. ADA macht es möglich, diese scheinbar inkompatiblen Ziele zur Deckung zu bringen, denn anders als jede andere Programmiersprache basiert sie auf modernen Konzepten des Software-Engineering.

Überdies ist ADA eine praxisorientierte Sprache, die berücksichtigt, daß heute die meisten Programme in einem Team von Programmierern erstellt werden. Diese eigentlich banale Feststellung weist jedoch ganz aktuell auf die zwei prinzipiellen Mängel der meisten Programmiersprachen hin: erstens, die Art, in der das menschliche Gehirn Probleme löst, ist in ADA durch den Einsatz von Abstraktionen nachgebildet. Sprachen wie zum Beispiel Pascal unterstützen die von oben nach unten strukturierte Programmierung. Versuche haben jedoch gezeigt, daß dies alleine nicht ausreicht.

Zweitens, wenn ein Programm durch ein Team erstellt wurde und nicht durch einen einzelnen Programmierer, so ist es wichtig, sicherzustellen, daß die Schnittstelle zwischen den Team-Mitgliedern effizient und zuverlässig ist. In der Regel wird ein Programm in relativ unabhängige Module aufgeteilt, die Schnittstellen zwischen den Modulen werden spezifiziert und die Implementation der Module auf das Team verteilt. Sicherzustellen, daß es nicht zu Überschneidungen der Arbeiten von Team-Mitgliedern kommt, erfordert einen beachtlichen Zeitaufwand. ADA wurde speziell daraufhin entwickelt, den Grad der Wahrscheinlichkeit, mit dem ein Fehler in einem Programmteil andere Einheiten des Programmes beeinträchtigt, zu minimieren.

Um alle durch das DOD spezifizierte Merkmale und Möglichkeiten bereitzustellen, ist ADA eine reiche, expressive Sprache. Einige ihrer Konzept-Bestandteile und formalen Eigenschaften werden dem neuen Anwender nicht sofort vertraut sein. Die Sprache ist jedoch weitgehend von Pascal abgeleitet und die Vertrautheit mit Pascal oder ähnlich gearteten Sprachen ist eine ideale Voraussetzung zum Lernen von ADA. Darüber hinaus sind die reservierten Begriffe in ADA im wesentlichen Vollängen-Wörter und weniger Abkürzungen, was das Lesen und Verstehen von Programmen vereinfacht.

Eines der wesentlichen Entwicklungsziele bei der Entwicklung von ADA war das Erbringen der neuesten Software-Engineering-Konzepte. Fortschritte in der Soft-

ware-Technik werden nicht so breit veröffentlicht und bereitwillig übernommen, wie vergleichbare Fortschritte in der Hardware-Technik. Daraus resultiert, daß Programmierer oft gezwungen werden, zwischen Techniken zu wählen, die jeweils einen exklusiven Ansatz zu bieten scheinen, dies aber in Wirklichkeit nicht tun.

In den vergangenen Jahren wurde den theoretischen Vorteilen von Top-down-Software-Konzepten viel Aufmerksamkeit geschenkt. Jedoch lassen sich die Software-Entwicklungskosten nur senken, wenn neue Programme unter Nutzung erprobter Module bereits vorhandener Programme erstellt werden. Natürlich kann eine „Neuentwicklung von Grund auf" kommerzielle Vorteile bieten. In jeweils ähnlicher Weise erfordern die meisten Programmiersprachen eine sequentielle Abarbeitung der verschiedenen Programm-Bestandteile, auch wenn zahlreiche Applikationen logische oder sogar physikalische Parallelität voraussetzen. Dank des großen Spektrums der Funktionselemente, der leistungsfähigen Modularität und der Abstraktionsmöglichkeiten lassen sich diese offensichtlich konträren Erfordernisse durch ADA in Einklang bringen.

Die meisten Programmiersprachen ermöglichen die Aufteilung eines Applikationsprogrammes in kleinere Teile, genannt Prozeduren oder Subroutinen. Gemäß des Konzeptes der Top-down-Entwicklung besteht ein ADA-Programm aus einer Sammlung von Blöcken, genannt „Programm-Einheiten". ADA ist jedoch vielseitiger als andere Sprachen, weil sie vier verschiedene Arten von Programm-Einheiten bietet:
- Subprograms,
- Packages,
- Tasks,
- Generics.

„Subprograms" entsprechen den Prozeduren und Funktionen anderer Sprachen während das „Package" ein wichtiges Funktionselement zur Unterstützung der Abstraktion und Modularität ist. Die „Task" ist die Basis-Einheit der Parallel-Verarbeitung. Tasks werden in logischer Form parallel abgearbeitet und kommunizieren mit anderen Tasks durch einen Mechanismus auf höherer Ebene, bekannt als „Rendezvous". „Generics" verbessern die Software-Produktivität durch die Möglichkeit, Algorithmen und Datenstrukturen auf einem höheren Abstraktionsniveau zu spezifizieren als andere Programm-Einheiten.

„Typing"

Ein anderes wesentliches Entwicklungsziel bei ADA war es, die Fehlererfassung so früh wie möglich einsetzen zu lassen, um die Kosten für die Korrektur zu minimieren. Wie Pascal unterstützt auch ADA den Compiler bei der automatischen Fehlererfassung durch Zuordnung eines „Type" zu jedem „Objekt". In ADA ist ein „Objekt" alles, was einen Wert annehmen kann und jedes „Objekt" muß vor seiner Benutzung vereinbart sein. In der Vereinbarung ist das „Objekt" mit einem „Type" verbunden, womit die erlaubten Werte und die Operationen definiert werden, die ausführbar sind. *Bild 1* zeigt die große Vielfalt von „Types", die in ADA verfügbar sind. Darin sind nicht nur vordefinierte „Types" wie INTEGER, BOOLEAN, CHARACTER und FLOAT, sondern auch neue Datentypen enthalten, die der Programmierer definiert hat.

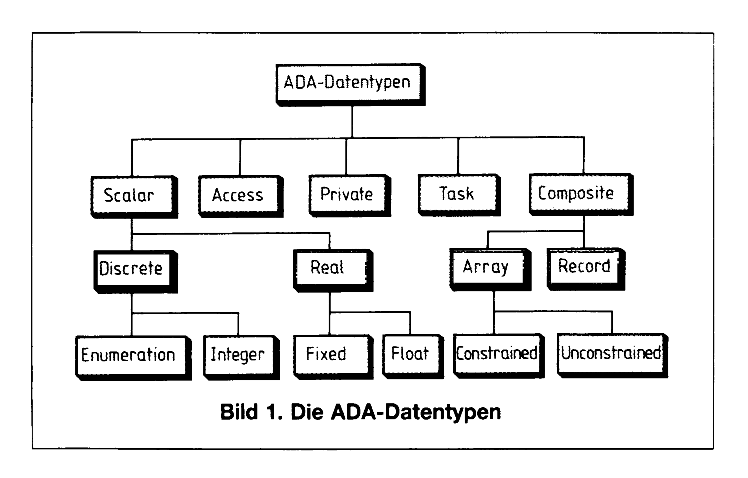


Bild 1. Die ADA-Datentypen

Der Compiler erzwingt rigoros einen konsistenten Einsatz der Datentypen. Ein Beispiel: wenn eine Variable A vom Typ INTEGER ist und eine Variable B vom Typ FLOAT, so würde der Versuch, A = B zu setzen, einen Compilierungsfehler in ADA erzeugen, weil die Gleichsetzung von Operatoren nur dann erfolgen kann, wenn die Objekte vom selben Datentyp sind. In „C" dagegen würde der Compiler die Variable A ohne weiteres in den entsprechenden Fließkomma-Typ umsetzen und keinen Fehler anzeigen, obwohl das Auftreten eines Durcheinanders bei den Datentypen vermutlich auf einen logischen Programmfehler hinweist.

Die Möglichkeiten, in ADA neue Datentypen zu definieren, bedeuten eine erhebliche Hilfe bei der Verbesserung der Verständlichkeit und Portierbarkeit von Programmen. Bei der Entwicklung der meisten anderen Sprachen war die Lesbarkeit der Programme nicht das Hauptanliegen. Zum Beispiel definiert folgende Vereinbarung einen neuen „Type-CHANNEL-STATUS", der die Werte (Zustände) DOWN, BUSY und FREE aufweisen kann:

```
type CHANNEL-STATUS is (DOWN, BUSY, FREE)
```

Ein variabler CURRENT-CHANNEL-Einsatz, definiert als Typ des CHANNEL-STATUS, wird nun auf die definierten Werte bzw. Zustände begrenzt sein und der Compiler würde jeden Versuch zurückweisen, andere Werte oder Zustände den Variablen zuzuweisen.

Der Vorteil, der sich aus der Definition neuer Datentypen ergibt, ist bei folgendem Ausschnitt ersichtlich:

```
  if CURRENT-CHANNEL-STATUS = FREE then
  ALLOCATE-MESSAGE-TO-CHANNEL;
elsif CHANNEL-NUMBER =< CHANNEL-LIMIT then
```

```
    TRY-NEXT-CHANNEL;
else
    REPORT-NO-FREE-CHANNELS;
end if;
```

Ein direkter Nutzen ist es, daß keine Kommentarzeilen erforderlich sind, um die auszuführenden Funktionen zu erläutern. Durch die Definition von Datentypen, die relevant für die jeweilige Task sind, kann der Programmierer sein Programm selbst dokumentierend machen. Ohne diese Möglichkeit müßte der Programmierer numerische Codes wie zum Beispiel „0" für einen freien Kanal und „1" für einen belegten Kanal usw. entsprechend zuordnen. Falls irgend jemand später das Programm nachvollziehen muß, so wäre die Bedeutung der speziellen Kanal-Codes nicht ohne weiteres ersichtlich.

In bezug auf die Programmausführung werden die numerischen Codes natürlich den Werten DOWN, BUSY und FREE zugeordnet, so wie sie oben definiert wurden, aber der wesentliche Unterschied ist, daß die tatsächliche Zuordnung durch den ADA-Compiler und nicht durch den Programmierer vorgenommen wird. Obwohl es der Hauptzweck einer Hochsprache ist, dem Programmierer das Denken in Begriffen seiner Applikation zu ermöglichen, fordern die meisten Sprachen dem Programmierer ab, einen erheblichen Zeitaufwand darin zu investieren, zu entscheiden, wie die Datenstrukturen zugeordnet werden sollen. ADA überträgt die Verantwortung für diese wichtige Aufgabe vom Programmierer auf den Compiler.

Abstraktionen und Pakete

Die Möglichkeit, neue Datentypen zu kreieren, ist nur ein Aspekt einer sehr viel leistungsfähigeren Technik, genannt „Abstraktion". Eine Abstraktion ist im wesentlichen eine vereinfachte Beschreibung eines Systems oder Subsystems. Die Beschreibung wird durch Konzentration auf die wesentlichen Merkmale, die der Anwender des Systems kennen muß, vereinfacht. Weniger wichtige Dinge bleiben im Hintergrund. Die Hervorhebung der Schlüsselfunktionen unterstützt insofern die Programmentwicklung, als das unwichtige Dinge nicht die Aufmerksamkeit des Programmierers in Anspruch nehmen. Genauso wichtig ist es, daß Details der Implementation anderer Programmteile verdeckt bleiben und somit die Zuverlässigkeit der Software insgesamt gesteigert wird.

Weit davon entfernt, ein neues Konzept zu sein, spielt die Abstraktion eine fundamentale Rolle im menschlichen Denken. Die meisten Dinge des Alltags wie Autos, Computer oder Telefone sind in solchem Sinne Abstraktionen, als das der Benutzer nicht wissen muß, wie und warum sie funktionieren, um sie zu gebrauchen. Alle Programmiersprachen beinhalten einen gewissen Grad an Abstraktion, aber bei den meisten Sprachen ist dies begrenzt auf den Einsatz von Unterprogrammen. Bei ADA wurde das Konzept der Abstraktion dadurch erweitert, daß die Implementation von Details eines Objektes oder einer Programm-Einheit vollständig aus anderen Einheiten möglich ist. Das Schlüsselfunktionselement in ADA zur Erzeugung von Abstraktionen wird „Paket" (Package) genannt.

Ein Paket ist eine Sammlung von Ressourcen, die von anderen Teilen des Programmes genutzt werden kann. Diese Ressourcen können konstante und variable Vereinbarungen, Typenvereinbarungen, Prozeduren, Funktionen, Tasks, Exceptions, Generic-Units oder andere Pakete beinhalten. Jedes Paket muß einen Bestandteil haben, der „Package-Specification" genannt wird und eine Vereinbarung über die für andere Programm-Einheiten verfügbaren Ressourcen darstellt. Darüber hinaus beinhalten die meisten Pakete eine zweite Komponente, genannt „Package-Body". Der „Body" beinhaltet die detaillierte Implementation jeder Prozedur oder Funktion, die in den Spezifikationen vereinbart ist.

Ein wichtiger Nutzen der Paketkonstruktion wird deutlich, wenn man die Implementation einer Warteschlange betrachtet. Warteschlangen sind in zahlreichen Applikationen wichtige Datengebilde und können auf verschiedene Weise implementiert werden, zum Beispiel als Verbindungslisten oder Kreis-Arrays. Wenn man sie als Abstraktion betrachtet, ist die Implementation einer Warteschlange bedeutungslos. Alles, was der Benutzer über die Warteschlange wissen muß, ist, welche Operationen ausgeführt werden können (ENTER, REMOVE) und welche Fehlerbedingungen auftreten können (QUEUE-FULL, QUEUE-EMPTY).

Bild 2 zeigt die Paket-Spezifikation eines ADA-Paketes, das es anderen Programm-Einheiten erlaubt, Zeichen-Warteschlangen zu implementieren. Die Spezifikation wird durch ein „Package-Body" vervollständigt, das im jeweiligen Falle die Prozeduren und Funktionen implementiert, die in der Spezifikation bezeichnet sind. Jede Programm-Einheit, die Zeichen-Warteschlangen manipulieren muß, kann QUEUE-PACKAGE benutzen, um dies zu tun, jedoch nur in bezug auf die Prozeduren und Funktionen, die in der Spezifikation definiert sind.

Während es ebenso möglich ist, eine einfache Prozedur zu schreiben, die eine Warteschlange behandelt, führt der Einsatz eines Paketes zu einer Verbesserung der Software-Zuverlässigkeit. Ein Prozedur-QUEUE-HANDLER macht die Implementation-Details einer Warteschlange für den Anwender der Prozedur sichtbar. Die Versuchung, die Codelänge oder die Ausführungs-

```
'package QUEUE_PACKAGE is
    procedure ENTER (C: in CHARACTER);
    function REMOVE return CHARACTER;
    function QUEUE_EMPTY return BOOLEAN;
    function QUEUE_FULL return BOOLEAN;
end QUEUE_PACKAGE;
```

Bild 2. Eine typische Paket-Spezifikation

zeit durch direkten Eingriff auf Maschinenebene in die Elemente der Warteschlange zu verringern, ist etwas, dem viele Programmierer nur schwer widerstehen können. Unglücklicherweise führen solche Abkürzungen häufig zu subtilen Fehlern. Ist zum Beispiel die Warteschlange als Kreis-Array implementiert, so fügt der Programmierer vielleicht ein neues Zeichen direkt in die Warteschlange ein, vergißt aber, den Zeiger zu verstellen, der das Ende der Schlange markiert. Resultat daraus ist, daß die nächste Einfügung die vorherige überschreiben wird.

Im Gegensatz dazu können andere Programm-Einheiten nicht direkt auf die Elemente der Schlange einwirken, wenn die Warteschlange in ein Package-Body implementiert ist. Die Warteschlange ist einzig und allein in bezug auf die Operationen definiert, die in ihr ausführbar sind und in bezug auf die Typen von Datenobjekten, die sich in ihr befinden. Darüber hinaus braucht nur der Package-Body verändert werden, wenn später entschieden wird, die Art, in der die Warteschlangen implementiert sind, zu ändern.

Software-Produktivität

Es wird geschätzt, daß Programmierer, die an komplexen Programmen arbeiten, im Durchschnitt fünf fehlerfreie Programmzeilen pro Tag erstellen können. Seit langem ist daher die Notwendigkeit erkannt, die Produktivität der Programmierer zu erhöhen. Eine Möglichkeit zur Steigerung der Produktivität ist die Verwendung von schon erstelltem Code, soweit dies möglich ist. Es ist klar, wenn ein Programm in logisch-strukturierte Module aufgeteilt werden kann, in dem Sinne, daß einige dieser Module direkt aus einer Bibliothek aufrufbar sind, und wenn die Programmiersprache die unabhängige Compilierung von Modulen zuläßt, so läßt sich die Zeit zur Erstellung eines Programms erheblich reduzieren.

Die meisten Programmiersprachen erlauben in der Tat die unabhängige Compilierung, aber dies ist im allgemeinen durch den Verlust von Zuverlässigkeit erkauft, da die Schnittstellen zwischen den Modulen nicht durch den Compiler geprüft werden. Im Gegensatz dazu macht es keinen Unterschied, ob ein ADA-Programm aus einer Zahl von compilierten Einheiten oder aus einem einzigen compilierten Code besteht. Der Grad, mit dem der Compiler das Programm überprüft, bleibt derselbe. Dieses Merkmal, bekannt als separate Compilierung, stellt die frühe Fehlererfassung sicher, egal ob das Programm nach einer Top-down- oder Bottom-up-Strategie erstellt wurde.

In den zurückliegenden Jahren wurden verschiedene fortschrittliche Entwicklungssprachen vorgestellt, die Möglichkeiten für die Erstellung von Software-Schablonen enthalten, die zur Herstellung von speziellen Programm-Einheiten anpaßbar sind. ADA ist die erste kommerzielle Sprache, die eine solche Möglichkeit beinhaltet, wie sie die „Generic-Program-Units" darstellen.

Ein Beispiel stellt in dieser Hinsicht das oben beschriebene Warteschlangen-Paket dar. Die Paket-Spezifikation in Bild 2 schließt eine Typ-Vereinbarung ein, die den Einsatz des „QUEUE-PACKAGE" auf Zeichen-Warteschlangen beschränkt. Die abstrakte Definition einer Warteschlange hängt jedoch nicht von dem Typ des Objektes ab, das in der Warteschlange enthalten ist. Es sollte möglich sein, in gleicher Weise Warteschlangen von Integern, Strings, Records oder anwenderdefinierten Datentypen zu behandeln. Mit ADA kann der Programmierer einen „Generic" schreiben, zum Beispiel QUEUE-TEMPLATE genannt, durch den die Art der Objekte nicht festgelegt wird.

Als wesentliche Eigenschaft erlauben es die „Generics", daß Sekundär-Merkmale aus den Programm-Einheiten ausgegliedert und als Parameter genutzt werden können. Nimmt man Warteschlangen als Beispiel, so sind primäre Merkmale, daß Elemente eingefügt oder entfernt werden können und daß die Meldungen QUEUE-IS-FULL und QUEUE-IS-EMPTY Boolesche Wertigkeiten haben. Die Größe der Warteschlange und die Objekte, die sich darin plazieren lassen, sind sekundäre Merkmale.

Nachdem QUEUE-TEMPLATE compiliert und in die Programm-Bibliothek aufgenommen ist, lassen sich zahlreiche verschiedene Unterprogramme oder Pakete automatisch durch die Bereitstellung des Generic mit den dazugehörigen Parametern erstellen. Merke: Generic-Parameter sind völlig verschieden von Subprogramm-Parametern. Die Parameter von Subprogrammen müssen Objekte sein und sie werden dazu benutzt, einen speziellen Aufruf desselben Subprogrammes zu parametrisieren. Generic-Formal-Parameter dagegen können Objekte, Typen oder auch Subprogramme sein und sie werden dazu genutzt, neue Programm-Einheiten zu erzeugen, die dann als selbständige Einheiten in die Programm-Bibliothek aufgenommen werden.

Der ADA-Compiler bietet eine umfassende Prüffunktion um sicherzustellen, daß die gelieferten Parameter denen entsprechen, die in der Generic vereinbart wurden. Die Generics von ADA sind den Macro-Erweiterungen oder Quellcode-Kopiertechniken, die in anderen Sprachen benutzt werden, um bereits vorhandenen Code zu modifizieren, aus diesem Grunde weit überlegen. Wenn Anwender eine Bibliothek aus erprobten Generics aufbauen, werden sie feststellen, daß eine zunehmende Zahl von Programm-Einheiten automatisch generierbar ist. Das reduziert natürlich den enormen Zeitaufwand, der bislang vergeudet wurde, um bereits vorhandene Arbeit zu duplizieren.

Es gibt noch eine Vielzahl anderer leistungsfähiger Merkmale von ADA, die in diesem kurzen Überblick nicht beschrieben werden können. Zum Beispiel bietet ADA eine umfassende Unterstützung für die Parallel-Programmierung durch die Verwendung von Tasks.

Weil „Tasking“ ein Bestandteil der Sprache ist, sind implementationsabhängige Ablaufaufrufe nicht erforderlich und die entsprechenden ADA-Programme deshalb portierbar. ADA ermöglicht dem Programmierer darüber hinaus Anweisungen in Form von „Pragmas“ und „Representation-Clauses“ an den Compiler zu geben, was ihn von der Notwendigkeit entbindet, an Schlüsselpunkten des Programms in Assembler zu programmieren.

Representation-Clauses werden dazu benutzt, bestimmte Eigenschaften des Programmes in bezug auf die Hardware zu beschreiben und festzulegen. Eine Applikation kann Sensoren oder andere periphere Hardware beinhalten, die physikalisch einer bestimmten Adresse zugeordnet sind. In diesen Fällen erlaubt ein ADA-Representation-Clause, wie zum Beispiel

```
for DOOR-SENSOR use at 16£4000
```

die Zuordnung des variablen DOOR-SENSOR zu der hexadezimalen Adresse £4000. Dieselbe Technik kann dazu benutzt werden, einer Task-Entry einen Hardware-Interrupt-Vektor zuzuordnen, so daß das Auftreten eines Interrupts einen Entry-Call der Task auslöst.

Pragmas stellen Instruktionen für den Compiler dar, in einer bestimmten Weise zu arbeiten. So kann es zum Beispiel erwünscht sein, daß die Ausführungszeit einer bestimmten Prozedur auf ein Minimum hin optimiert wird. ADA beinhaltet ein vordefiniertes Pragma namens OPTIMISE (...), das mit einem der beiden Parameter SPACE oder TIME eingesetzt wird. Zum Beispiel:

```
procedure FAST-SEARCH is
   pragma OPTIMISE (TIME);
begin
---
end FAST-SEARCH
```

– vereinbart eine Prozedur FAST-SEARCH und informiert den Compiler, daß eine minimale Ausführungszeit das wichtigste Kriterium ist.

Die VADS-Umgebung

Obwohl ADA theoretisch auf jede Zielmaschine portiert werden kann, gibt es keinen Zweifel, daß einige Architekturen besser hierfür geeignet sind als andere. ADA wurde entwickelt, damit Programmierer mit einem Maximum an Effizienz und einem Minimum von Fehlern arbeiten können. Die Sprache stellt deshalb hohe Anforderungen an den Compiler, der eine viel umfangreichere Arbeit zu verrichten hat als Compiler für andere Sprachen. Die offizielle Zulassungs-Prozedur beinhaltet keine Messung der Leistungsfähigkeit eines Compilers. Es wird lediglich geprüft, ob alle ADA-Regeln eingehalten werden. Deshalb kann es signifikante Leistungsunterschiede zwischen verschiedenen zugelassenen Compilern geben. Ganz besonders wichtig ist, daß der Zielprozessor auf einer Architektur basiert, die eine effiziente Compilierung unterstützt. In dieser Hinsicht stellt die vollständige symmetrische Architektur der 32000-Serie einen wesentlichen Vorteil gegenüber anderen 32-Bit-Mikroprozessoren dar, weil sie einen compilierten Code hervorbringt, der vielfach genauso kompakt ist wie Assembler-Code. Obwohl der Compiler von größter Wichtigkeit ist, ist er, nicht das einzig erforderliche Software-Entwicklungszwerkzeug. Die meisten der Editoren, Linker, Loader und ähnliche Werkzeuge, die heute in Gebrauch sind, sind nicht als Teil einer integrierten Entwicklungsumgebung geschaffen worden. Die Aufgabe, sicherzustellen, daß diese Werkzeuge in jeweils gegenseitig kompatibler Weise arbeiten, obliegt dem Anwender. Im Hinblick darauf, daß zahlreiche Software-Entwicklungsprobleme durch den Mangel an wirklich integrierten Programmier-Werkzeugen verursacht oder verschärft werden, hat das DOD darüber hinaus eine integrierte ADA-Programmier-Unterstützungsumgebung (APSE) spezifiziert.

Auf der derzeitigen Entwicklungsstufe von ADA ist noch kein umfassendes APSE verfügbar, aber die meisten Hersteller von ADA-Compilern können APSE-Subsets liefern. Nach der Evaluierung verschiedener kommerziell verfügbarer ADA-Produkte hat National Semiconductor das Verdix-ADA-Entwicklungssystem (VADS) für die Serie 32000 ausgewählt. Verdix Corporation, mit Sitz in Washington und Oregon, ist 1982 zur Entwicklung von Mehrebenen-Sicherheits-Computer-Netzwerksystemen gegründet worden. Der erste ADA-Compiler dieses Unternehmens für VAX/Unix-Systeme wurde 1984 zugelassen. Das VADS-System ist in hohem Maße anerkannt und es gibt dafür zahlreiche Endanwender, einschließlich Hughes, Rockwell, TRW, Siemens

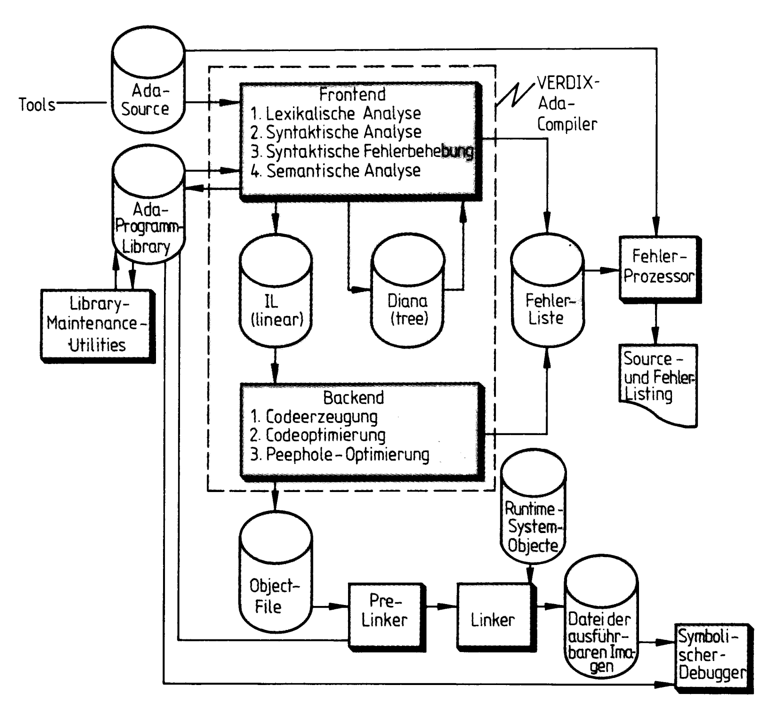


Bild 3. Das ADA-Entwicklungssystem von Verdix in Block-Darstellung

und Tektronix. Darüber hinaus hat die Firma DEC den Verdix-Compiler als bevorzugten ADA-Compiler unter ULTRIX ausgewählt.

Bild 3 zeigt die Möglichkeiten, die VADS bietet. Das Herzstück des Systems ist ein Hochleistungs-ADA-Compiler, der vollständig dem ANSI/MIL-STD-1815A-Standard entspricht. Das komplette System beinhaltet darüber hinaus einen Debugger, ein ADA-Runtime-System (Echtzeitkern) und Programm-Bibliotheks-Hilfen.

Der Compiler kann ADA-Quellenanweisungen mit einer Rate von 1000 pro Minute auf einem Computer mit virtuellem Speicher und einer Leistung von 1 MIPS verarbeiten, während bereits compilierte ADA-Module mit einer Rate von 20 000 Anweisungen pro Minute ablaufen. Weil das Lernen von ADA ein bißchen mehr Zeit in Anspruch nimmt als weniger leistungsfähige Sprachen, ist der Compiler ausgesprochen anwenderfreundlich gestaltet. So beschreiben zum Beispiel Fehlermeldungen nicht nur die Art des Fehlers, sie weisen den Anwender darüber hinaus noch auf die entsprechende Stelle im ADA-Handbuch hin, wo er detaillierte Erklärungen findet.

Nach dem Compiler ist zweifellos ein guter symbolischer Debugger das wichtigste Entwicklungswerkzeug. Weil der VADS-Debugger als Teil eines integrierten Entwicklungssystems konzipiert ist, wird hier dieselbe separate Compilierungsinformation erstellt und durch den Compiler benutzt. Das bedeutet, daß ablauffähiger Code keine symbolischen Informationen enthält und nicht für Zwecke der Fehlersuche modifiziert ist.

Der Debugger ist vollständig symbolisch. Er ermöglicht dem Anwender den Zugriff auf ADA-Variable mittels Namen und jederzeit den direkten Zugriff auf den ADA-Quellencode. Die umfassenden Debug-Möglichkeiten schließen bedingte Unterbrechungspunkte und Einzelschritt-Ablauf ein. Compilierte ADA-Programme können darüber hinaus auf ein Zielsystem der 32000-Serie geladen werden, während die Fehlersuche auf einem VAX-Host-Computer stattfindet.

Mit dem zusätzlichen Satz von Programm-Bibliothek-Hilfen und einem Echtzeit-System, das Tasking, Unterbrechungssteuerung und Ein-/Ausgangs-Operationen unterstützt, bietet das VADS-System die wesentlichen Werkzeuge für die Entwicklung von effizienten und zuverlässigen ADA-Programmen. Natürlich findet eine stetige Weiterentwicklung statt. Geplante Verbesserungen umfassen die Optimierung der vorhandenen Werkzeuge ebenso wie die Entwicklung neuer Komponenten, zum Beispiel Werkzeuge für die Quellensteuerung.

Die ADA-Fähigkeit stellt einen wichtigen Zusatz für die 32000-Familie dar. Für militärische Applikationen bietet National Semiconductor somit ein komplettes Paket, angefangen von der ADA-Software bis zu strahlungsresistenter Hardware. In einem größeren Kontext ermöglicht ein integriertes Entwicklungssystem den nichtmilitärischen Anwendern den Zugang zu der steigenden Zahl von Unternehmen, die herausgefunden haben, daß ADA eine praktische Lösung für das Problem der eskalierenden Software-Herstellungs- und -Wartungskosten darstellt.

32000-Universitäts-Programm

National Semiconductor entwickelte ein spezielles Programm, das Studenten aus vielen Fachrichtungen und Hochschulen anspricht: das „Serie-32000-Universitäts-Programm". Dieses Programm ist voll in das 32000-Familienkonzept eingebunden und ermöglicht es den Studenten, mit der gesamten Produktpalette vertraut zu werden. Das Universitäts-Programm ist dabei nicht nur für Universitäten gedacht, sondern für alle staatlich anerkannten Hochschulen, die zum überwiegenden Teil Ausbildung betreiben (im Gegensatz zu denen, die sich primär der Forschung widmen). Teilnehmen können Universitäten, Technische Universitäten, Fachhochschulen und gemeinnützige Weiterbildungsinstitute. Das Programm spricht Studenten aus Fachgebieten wie Informatik, EDV, Elektrotechnik, Nachrichtentechnik, Medizin, Naturwissenschaften usw. an. Projekte für kommerzielle Zwecke werden nicht unterstützt. National Semiconductor bietet fast alle Produkte der Familie zu günstigen Preisen im Rahmen dieses Programmes an: Bauelemente, Entwicklungs-Werkzeuge, Entwicklungs-Software, Betriebssysteme und spezielle Ausbildungskurse.

Wie Anwender aus der Industrie möchten auch die Hochschulen zuerst die 32000-Familie evaluieren. Mit anderen Worten, sie wollen in der Praxis prüfen, was die Familie in bezug auf Hardware und Software bietet. Natürlich sollte dieser Einstieg so kostengünstig wie möglich sein. Ein Punkt, der für Hochschulen oftmals noch wichtiger ist als für die Industrie, da die Ausbildungsmittel begrenzt sind. Aus diesem Grund stellte National Semiconductor 1982 erstmals auf dem Markt das „University Kit" vor. Dieses Angebot wurde 1985 erweitert und umfaßt z. Z. vier verschiedene Pakete. Jedes Paket besteht aus zwei komplett bestückten DB32000-Entwicklungsplatinen, zehn Instruction-Set-Reference-Manuals, zehn 32000-Datenbüchern und, nach freier Wahl, entweder einem Pascal- oder C-Compiler-Cross-Software-Entwicklungspaket, das auf DEC/VAX unter VMS (Pascal) oder Unix (C) läuft. Jedes Paket wird zu einem Preis angeboten, der ca. 15 % des normalen Verkaufspreises der einzelnen Bestandteile des Pakets darstellt. Die verschiedenen Teile des Pakets können natürlich auch einzeln (im Rahmen des Universitäts-Programms) zu Preisen, die nur die Materialkosten decken, erworben werden.

Dank der „University Kits" können die Studenten erste Erfahrungen mit dem Chip-Satz und einer Hochsprache machen, die auf der 32000-Familie verfügbar ist. Sollte die Hochschule die Basis der Serie 32000 erweitern wollen, so kann sie aus der gesamten Familie der Produkte wählen, was für ihre individuelle Ausbildungsaufgabe benötigt wird. Dadurch wird der Lieferumfang an die einzelnen Bedürfnisse angepaßt. Für Studenten, die intensiv mit der Serie 32000 arbeiten, sind In-System-Emulatoren, Betriebssysteme oder sogar auf der Serie 32000 basierende Entwicklungssysteme von Interesse. In diesem Fall bietet das Universitäts-Programm: ISE16- oder ISE32-In-System-Emulatoren, Genix-4.1- oder Genix-4.2-Betriebssysteme als Source, EXEC Real-Time Executive als Kernel, SYS32- oder VR32-Entwicklungssysteme und ICM-3216-Integrated-Computer-Module.

Lee Tapper

Konzept für ein kosteneffektives Systeminterface

Platinencomputer kommt ohne Backplane aus

Bei der ICM-Prozessorfamilie handelt es sich um UNIX-Maschinen und Peripherieeinheiten, die für OEM-Anwendungen konzipiert sind und auf den Prozessorchips der Serie 32000 von National Semiconductor basieren. Nachdem bereits seit einiger Zeit Produkte dieser Familie ihre Praxistauglichkeit nachweisen können, soll hier ein neues Mitglied, das System ICM3232, vorgestellt werden. Diese Platine besteht aus einer Hochgeschwindigkeits-32-Bit-CPU, die mit einem E/A-Prozessor und einem Speicher mit Hilfe eines schnellen 32-Bit-Busses mit der Bezeichnung „MaxiBus" verbunden ist.

Bild 1 zeigt die Blockschaltung des Systems ICM3232. Die E/A-Leitungen kommen zum E/A-Prozessor (IOP) von den auf der Platine befindlichen Peripherieeinheiten sowie über einen 16-Bit-E/A-Bus, der die Bezeichnung „MiniBus" trägt. Der E/A-Prozessor entlastet den Zentralprozessor, indem er E/A-Informationen vorverarbeitet. Weil das ICM-System für Ansprüche des OEM-Marktes konzipiert ist, ist es besonders interessant, die Interface-Probleme zu zusätzlich anzuschließender Hardware zu lösen.

Zusätzliche Systemhardware wird an das System heute über Busse angeschlossen. In den meisten derzeit üblichen busorientierten Systemen ist das Businterface ein anspruchsvolles und teures Stück Hardware. Systeme wie z. B. Multibus II erfordern großen Aufwand beim Anschluß von Hardware an den Systembus oder den privaten Speicherbus. Die Kosten eines Businterfaces sind allerdings akzeptabel, weil dies letztendlich zum Anschluß an einen relativ teuren 32-Bit-Prozessor dient.

Obwohl der OEM-Benutzer Anpassungsprobleme an den Systembus lösen muß, kommt es relativ selten vor, daß er eine Schnittstelle zu einem 32-Bit-Prozessor benötigt. In diesem Fall kann der Preis für die Bus-Interface-Hardware schon eine merkliche Höhe annehmen. Als Beispiel stelle man sich einen Benutzer vor, der einen 8-Bit-A/D-Umsetzer an das System anschließen möchte. Dieser Fall kann eintreten, wenn ein moderner leistungsfähiger Prozessor an existierende Steuerhardware anzuschließen ist.

32-Bit-Busse oft zu leistungsfähig

Bei den meisten 32-Bit-Systemen muß der Anwender seine Systemerweiterungen an einen Bus anschließen, der eigentlich für einen Hochgeschwindigkeits-Prozessor konzipiert ist. Dies kann dazu führen, daß der Hard-

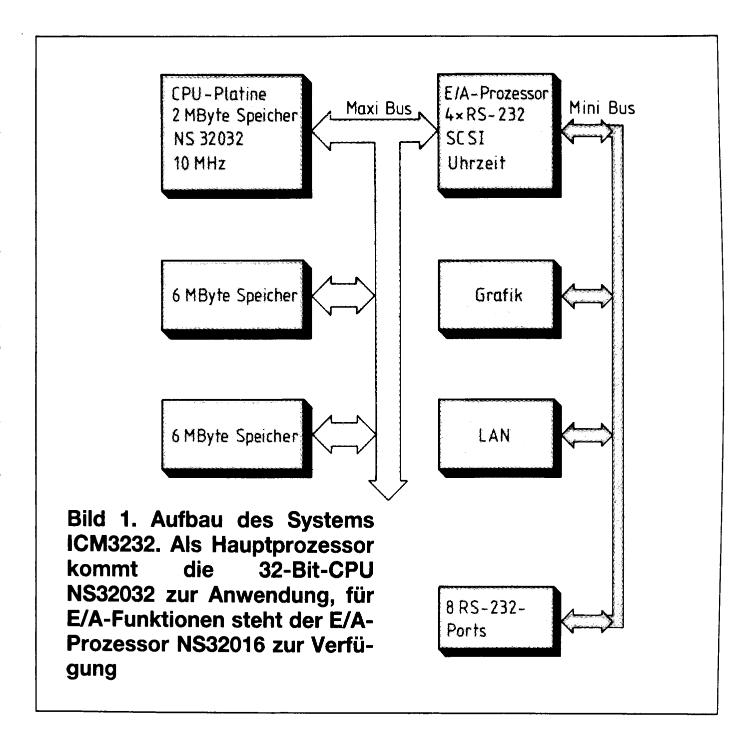


Bild 1. Aufbau des Systems ICM3232. Als Hauptprozessor kommt die 32-Bit-CPU NS32032 zur Anwendung, für E/A-Funktionen steht der E/A-Prozessor NS32016 zur Verfügung

ware-Aufwand für das Interface 5...10mal soviel wie die anzuschließende Schaltung kostet.

Die Computermodule der ICM-Familie versuchen auf zwei verschiedenen Wegen, hier eine wirtschaftliche Lösung zu bieten. Zuerst wurde der Aufwand für den eigentlichen Bus so stark verringert, indem viele der mechanischen Elemente, die üblicherweise dazu notwendig sind, entfielen. Zweitens ist das System so strukturiert, daß es eine ganze Hierarchie von Anschlußmöglichkeiten mit unterschiedlichem Preis-Leistungsverhältnis bietet. Wenn der Benutzer ein preiswertes langsames Bauelement anschließen möchte, muß er nicht die Hardware benutzen, die für einen 32-Bit-Mikroprozessor mit 10 MHz Taktfrequenz konzipiert war.

Ein wesentlicher Teil der Systemkosten eines Busses liegt im Kartenträger sowie der Backplane (oder Motherboard). Beim ICM-System (*Bild 2*) fehlen solche Elemente. Bei der ICM-Karte handelt es sich um eine Europakarte doppelter Breite und dreifacher Länge. Auf zwei Seiten befinden sich 96polige DIN-Stecker. Diese werden zum Anschluß von zwei Bussen des ICM-Systems benutzt.

Die DIN-Stecker sind als Federleiste auf der Bauelementeseite der Platine und als Steckerleiste auf der Lötseite ausgelegt. Damit ist es möglich, daß mehrere Platinen übereinandergesteckt werden können. Die anderen beiden Seiten der Platine bieten Platz für zwei Grundstecker zum Anschluß der Versorgungsspannung und einer Metallplatte, das als E/A-Anschlußfläche benutzt werden kann. Dieser Aufbau sorgt dafür, daß die Platine auf allen vier Seiten mechanisch unterstützt wird. Weil die Systembusse direkt durch den Stapel der durch die DIN-Stecker übereinandergestapelten Platinen fließt, gibt es keine Notwendigkeit für einen Kartenträger oder andere mechanische Befestigungsmittel. Mehrere übereinander gestapelte Karten können auch eine OEM-Konfiguration für ein sehr kleines System darstellen, wo kein Platz für Befestigungsrahmen oder ähnlicher Hardware ist. Dies kann wichtig sein, wenn das System nur begrenzten Platz zur Verfügung hat.

Bild 2. Die ICM-Platine benötigt keine zusätzlichen mechanischen Befestigungseinrichtungen. Die Busstecker für den MiniBus und MaxiBus erlauben ein Stapeln mehrerer Platinen

Zwei Busse für die Systemerweiterung

Die beiden Busse, die die DIN-Stecker verbinden, werden MaxiBus und MiniBus genannt. Beim MaxiBus handelt es sich um einen Hochgeschwindigkeits-Speicherbus, der in erster Linie zur Benutzung durch die CPU gedacht ist. Im Gegensatz dazu ist der MiniBus zum Anschluß der E/A-Leitungen vorgesehen. Daher ist der MiniBus der interessantere zum Anschluß zusätzlicher Hardware.

Der MiniBus ist ein 16-Bit-Bus, der insbesondere für E/A-Aufgaben konzipiert ist. Eine Breite von 16 Bit wurde gewählt, weil dies optimal zu den meisten E/A-Einheiten paßt. Peripherieeinheiten arbeiten üblicherweise mit 8 oder 16 Bit und nur selten mit einer Breite von 32 Bit. Daher legte man den MiniBus für 16 Bit aus, wobei man im Gegensatz zu einem 32-Bit-Bus einen sehr großen Aufwand und damit Kosten einsparte. Der MiniBus ist ein synchroner Multi-Master-Bus, an dem sich bis zu acht Master anschließen lassen. Die Master kommunizieren miteinander mit Hilfe von Datentransfers und virtuellen Interrupts. Jeder Busmaster besteht üblicherweise aus einer E/A-Controllerkarte, die von einem Mikroprozessor NS32016 gesteuert wird. Die Schnittstelle zwischen dem Prozessor und dem MiniBus wird von einem LSI-Baustein realisiert, der die Bezeichnung „MiniBus-Interfacecontroller“ (MBIC) trägt. Eine Hälfte der MBIC-Anschlüsse paßt direkt an den Bus des NS32016, der Rest paßt direkt an den MiniBus. Um auf Daten vom MiniBus aus zuzugreifen, liest oder schreibt die CPU in einem 4-MByte-Bereich des Speichers, der als MiniBus-Adreßbereich decodiert wird. Virtuelle Interrupts werden ausgesendet, indem ebenfalls auf kleine Bereiche dieser 4-MByte-Zone zugegriffen wird.

Der einfachste Zyklus auf dem MiniBus ist der 8-Bit-E/A-Zyklus. Dieser muß von einem MBIC-Baustein initialisiert werden, als Zieleinheit des E/A-Zyklus muß allerdings nicht dieser MBIC dienen. Der 8-Bit-E/A-Zyklus erlaubt den direkten Anschluß von unintelligenten Peripherieeinheiten an den MiniBus. So ist zum Beispiel der direkte Anschluß des 8-Bit-A/D-Umsetzers, der bereits früher erwähnt wurde, leicht möglich. Das Ablaufdiagramm der zeitlichen Steuerung für den 8-Bit-E/A-Zyklus zeigt *Bild 3*. Hierbei wird eine 8-Bit-Portadresse auf 8 der 16 Adreß-/Daten-Leitungen ausgesendet. Die Daten werden auf den anderen 8 Leitungen entweder gesendet oder empfangen. Weil die Adressen parallel ausgegeben werden, sind weder Adreßlatches oder Demultiplexer erforderlich, um das Interface aufzubauen. Die Schaltung für eine 8-Bit-E/A-Einheit zeigt *Bild 4*. Das Gatter mit den drei Eingängen, das am A/D-Umsetzerchip direkt angeschlossen ist, wird zur Decodierung des 8-Bit-E/A-Status vom MiniBus-Funktions-

code herangezogen. Die MiniBus-Leitungen A16 und A17 arbeiten sowohl als Schreib- als auch Lese-Leitungen im 8-Bit-E/A-Modus.

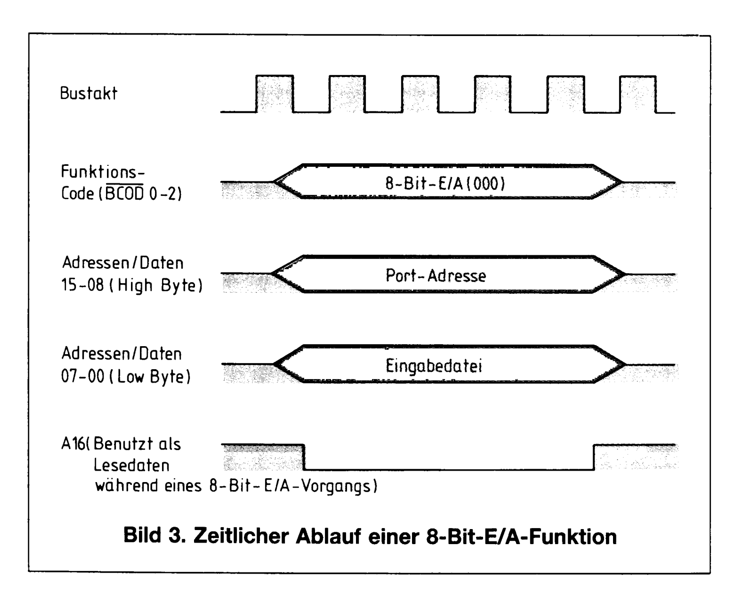


Bild 3. Zeitlicher Ablauf einer 8-Bit-E/A-Funktion

8-Bit-E/A-Funktionen sind die billigste Möglichkeit und bieten die geringste Leistung. Einen Schritt weiter geht die MiniBus-E/A-Funktion, die mit dem MBIC-Baustein arbeitet. Diese decodiert die Adressen des NS32016 und leitet einen Teil davon auf den MiniBus. Wenn ein bestimmter E/A-Controller aus dem Speicher eines anderen MiniBus-Teilnehmers lesen will, erzeugt das Steuerprogramm die entsprechende Adresse. Der MBIC des Controllers decodiert diese Adresse als eine MBIC-Adresse, übernimmt den Bus und sucht die Zieleinheit als Slave.

MBIC vereinfacht Schnittstelle

Die Verwendung des MBIC-Bausteins gestaltet die Schnittstelle zum MiniBus sehr einfach, obwohl dieser einige Funktionen bietet. Um mit dem MiniBus zu arbeiten, steht lediglich die CPU NS32016 zur Verfügung, mit dem der Hardware-Designer sich vertraut machen muß. Alle Funktionen des MiniBus-Protokolls werden vom MBIC bearbeitet und können zum größten Teil vergessen werden. Der MBIC enthält MiniBus-Treiber, so daß sich insgesamt eine sehr preiswerte Lösung für ein leistungsfähiges E/A-Bussystem ergibt. Als einzige zusätzliche Logik ist eine Schaltung zur Taktsynchronisation mit den Steuerleitungen erforderlich, die das Gate-Array benutzt, wenn auf lokalen Speicher zugegriffen wird. Die meisten E/A-Controller benutzen die String-Move-Instruktion des NS32016 zum Transfer von Daten über den MiniBus. In den Zellen, in denen die String-Move-Befehle nicht schnell genug sind, lassen sich höhere Transfer-Frequenzen erreichen, wenn anstelle des Bausteins NS32016 eine DMA-Zustands-Maschine tritt.

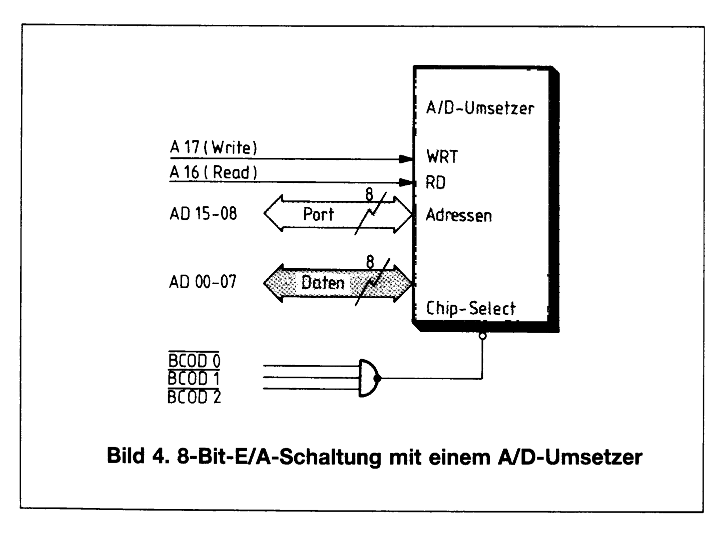


Bild 4. 8-Bit-E/A-Schaltung mit einem A/D-Umsetzer

Am oberen Ende der Preis-Leistungs-Kurve findet man den MaxiBus. Hierbei handelt es sich um einen Hochgeschwindigkeits-32-Bit-Speicherbus, den der Prozessor benutzt, um auf den Speicher zuzugreifen. Der Bus wird gemeinsam mit einem System-E/A-Prozessor (IOP) benutzt. Der Prozessor NS32032 kann über den MaxiBus bei einer Taktfrequenz von 10 MHz ohne Wartezyklen arbeiten. Es handelt sich beim MaxiBus um ein DRAM-Interface mit begrenzten Arbitrierungsmöglichkeiten. Die maximale Anzahl der Devices, die der Bus unterstützen kann, liegt bei vier. Die Bus-Übertragungsfrequenzen liegen über 10 MBit/s. Wie bei anderen schnellen 32-Bit-Bussen ist auch hier das Interface sehr komplex und teuer, so daß nur Einheiten, die wirklich diese großen Bandbreiten benötigen, über diesen Bus angeschlossen werden sollten. Beispiele dafür sind Hochgeschwindigkeits-Peripherieeinheiten wie eine Grafikplatine oder ein Array-Prozessor.

Optimales Preis-Leistungsverhältnis

Die Systemphilosophie der ICM-Familie befaßt sich in erster Linie mit dem Preis-Leistungsverhältnis. Dies wurde auch beim Konzept des Systembusses berücksichtigt. Wenn man Einheiten mit geringer Bandbreite anschließt, reicht die 8-Bit-E/A-Schnittstelle aus. Damit steht ein sehr preiswertes Interface zur Verfügung. Höhere Leistung für Einheiten wie z. B. Grafik-Zusatzmodule oder LAN-Karten lassen sich mit Hilfe des MBIC erreichen. Auf dem oberen Ende steht der MaxiBus zur Verfügung.

Charles Rennolet

Platinencomputer als Basis für Workstations

Zur Konzipierung einer Workstation für ein Forschungslabor sollte ein Platinencomputer ausgewählt werden. Die wichtigsten Aufgaben, die in dieser Applikation anfallen, sind beispielsweise das Erstellen technischer Reports, Hardware- und Software-Entwicklung mit Schwerpunkt auf der Softwareseite. Die Benutzer sind Elektroingenieure, Physiker, Mathematiker und Computer-Ingenieure. Randbedingungen sind: möglichst niedriger Preis, das Betriebssystem Unix, bei dem Zugriff auf Quellencode und Text-Formatierungseinrichtungen möglich ist, sowie das gleichzeitige Abarbeiten mehrerer Anwendungsprogramme. Als Lösung ergab sich die Verwendung des Boards ICM3216.

Als Problem bei den vorhandenen Systemen (mit der CPU 68000) hatte sich herausgestellt, daß bei starker Belastung (vier Tasks gleichzeitig) die Leistung nicht mehr ausreichte. Eine Möglichkeit wäre der Kauf einer VAX gewesen, allerdings war sie, neben anderen Problemen, einfach viel zu teuer.

Inzwischen steht auf dem Markt eine brauchbare Alternative zur Verfügung: der Platinencomputer ICM3216 von National Semiconductor. Dieser läuft unter Unix (AT&T System V, Version 2.2), das die Textformatierungs- und Quellen-Code-Steuereinrichtungen besitzt, die für diese Anwendung erforderlich ist, und die Sprache C anstandslos verarbeitet. Jedes System ICM3216 kann bis zu vier Benutzer unterstützen. Darüber hinaus laufen alle Programme, auch wenn sie noch so anspruchsvoll sind. Dabei ist der Preis überraschend niedrig.

Die Platine ICM3216 besteht aus zwei übereinander gestapelten Einheiten von etwas größerer Tiefe, aber gleicher Länge wie eine VMEbus-Platine doppelter Höhe. Es gibt keinen Bus. Bei einer Platine handelt es sich um das CPU-Board mit dem 32000-Chipsatz, nämlich MPU32016, MMU32082, FPU32081, Interrupt- und Zeitsteuereinheit. Die Platine verfügt auch über einen Uhren-Kalender-Baustein, vier serielle Ports, ein Centronics-Druckerport, Zeitgeber und Zähler (es handelt sich hierbei um zwei sehr elegante Multifunktions-Bausteine von Signetics), ein SCSI-Port, das mit einer CPU Z80B und einem SCSI-Chip von NCR aufgebaut ist, einem Erweiterungsstecker für den lokalen Bus, ROM und einem Minibus-Anschluß, der später noch erläutert wird. Bei der anderen Platine handelt es sich um eine Speichererweiterung auf der lokalen Bus-Erweiterungsleitung, der bis zu 4 MByte Kapazität (256-KBit-Chips) enthält. Zwei Speicherplatinen lassen sich übereinander stecken, so daß insgesamt eine Kapazität von 8 MByte erreichbar ist. Die Platinen lassen sich mit Hilfe von DIN-Steckern miteinander verbinden. An den vier Seiten der CPU-Platine befinden sich Anschlußpfosten für die Stromversorgung aus einer 5-V-Quelle. Andere Spannungsversorgungen liegen an Standardanschlüssen auf der hinteren Kante.

Wenn man diese Platine zum ersten Mal sieht, kann man sie leicht mit einem VMEbus-System verwechseln, das mit einer Speicherplatine aufgestockt wurde. Die Anschlüsse für die seriellen Ports (zwei Telefonstecker und zwei DB-25-Stecker) sowie das Parallelport (ein DB-25-Stecker) liegen an der Vorderseite.

Das interessanteste an diesen Platinen ist der Preis: etwa 2000 $ für die Hardware (mit 1 MByte Speicher) und weniger als 300 $ für die Unix-Lizenz. In großen Stückzahlen kostet die Konfiguration sogar nur noch den halben Preis. Obwohl zum Betrieb noch ein Plattenlaufwerk, Controller, Bandlaufwerk und andere Dinge hinzuzufügen sind, ist das ein außerordentlich preiswürdiges Konzept. Die Integration von Controller, Platten- und Band-Laufwerk in dieses System ist sehr stark vereinfacht, denn es ist ein SCSI-Interface vorhanden, bei dem ein Standard-Protokoll-Chip die Steuerung übernimmt.

Genutzt wird System 1, das über einen SCSI-Plattencontroller von Emulex verfügt sowie ein kleines (3½ Zoll) ST-506-Laufwerk mit 20 MByte Kapazität. Dies reicht allerdings nicht aus, um das vollständige Unix mit allem Drumherum unterzubringen, so daß das Quellencode-Steuersystem und die Text-Formatierungssoftware weggelassen wurden. Die Anforderungen an die Stromversorgung sind nicht sehr hoch: 5 V bei maximal

8 A und ±12 V mit sehr geringer Leistungsaufnahme. Ein relativ kleines geschaltetes Stromversorgungsgerät reicht aus, um dies zur Verfügung zu stellen. Die ganze Einheit ist in einem Aluminiumgehäuse untergebracht, das nicht größer als eine Aktentasche ist; alles zusammen wiegt etwa 5 kg. Damit ist es leichter als so mancher „portable“ Computer.

Soviel zur Hardware. Nach Anschluß eines üblichen VT-100-ähnlichen Terminals und Einschalten des Systems meldete sich der Monitor. Wenige Tastenbetätigungen genügten, um in den Single-User-Mode von Unix zu gelangen. Weitere wenige Tastenbetätigungen genügten, um Multi-User-Unix zum Laufen zu bringen. Bei allen diesen ersten „Gehversuchen“ stellte sich heraus, daß obwohl der Boot- und Abschalt-Vorgang die schwierigsten Betriebszustände sind, keine schwerwiegenden Fehler auftraten. Alles, was in der Versuchsphase ausprobiert wurde, arbeitete in der üblichen Weise. Auffällig war nur, daß es ein wenig schnell funktionierte.

Aus Zeitgründen wurden zunächst nicht alle Unix-Kommandos erprobt. Mehrere Benutzer, die mehr „spielerisch“ mit dem System arbeiteten, fanden keine Probleme und waren beeindruckt von der Arbeitsgeschwindigkeit sowie der einfachen Bedienbarkeit. Die Unix-on-Line-Help-Einrichtung in Form des „Man“-Kommandos ist wirklich eine Hilfe, denn im Gegensatz zu vielen anderen Systemen unterstützt es alle vorhandenen Kommandos. Mit dessen Hilfe wurden einige Programme erprobt, um für unterschiedliche Umgebungsbedingungen erste Ergebnisse zu haben. Auch hier traten keine Probleme auf.

```
        FIB.C
        This benchmark tests, roughly, the basic speed
        at which stack operations take place, which, in
        most cases is just memory access speed.

#define NTIMES 10
#define NUMBER 24

main()
{
  int i;
  unsignet value, fib();

  printf("/n%d iterations: ", NTIMES);

  for (i = 1; i <= NTIMES; i++)
    value = fib(NUMBER);
  printf("\nfibonacci(%d) = %u", NUMBER, value);
  exit(0);
}

unsigned fib(x)
int x;
{
  if (x > 2) return(fib(x - 1) + fib(x - 2));
  else return(1);
}

        SIEVE.C
        SIEVE OF ERATOSTHENES -- standard benchmark,
        included because data is available for it for
        almost every system around.

#define true 1
#define false 0
#define size 8190

  char flags(size+1);

main()
{
  int i, prime, k, count, iter;
  printf("\n10 iterations");
  for (iter = 0; iter <= 10; iter++)
}
    count = 0;
    for (i = 0; i <= size; i++)
      flags[i] = true;
    for (i = 0; i <= size; i++)
      if (flags[i])
      {
        prime = i + 1 + 3;
        for (k = i + prime; k <= size; k += prime)
          flags[k] = false;
        count++;
      }
  }
  printf("\n%d primes.", count);
}

        FPBENCH.C -- a fairly simple test of some
        elementary floating point capability

#define CONST1 3.141597E04
#define CONST2 1.78E02
#define COUNT 10000

main()
{
  double a,b,c;
  int i;

  a = CONST1
  b = CONST2
  for(i = 0; i < COUNT; i-+)
  {
    c = a * b;
    c = a / b;
    c = a * b;
    c = a / b;
    c = a * b;
    c = a / b;
    c = a * b;
    c = a / b;
    c = a * b;
    c = a / b;
    c = a * b;
    c = a / b;
  }
}
```

Bild 1. Verwendete Benchmarks

Bevor man zu Leistungsvergleichen übergeht, sollten hier noch einige Punkte genannt sein: Bisher gibt es keine vergleichbare Hardware in dieser Preisklasse, auf der auch Unix läuft. Ein typisches System, das aus dem Platinensatz ICM3216, einem Laufwerks-Controller, einem Platten- und Band-Laufwerk und verschiedenen anderen Teilen besteht, kann leicht für etwa 5000 $ zusammengestellt werden. Die Kosten für jedes andere der getesteten Systeme liegt über 10 000 $, bei einigen kann man sogar von einem Preis über 30 000 $ ausgehen.

Bevor auf die Resultate und Vergleiche näher eingegangen wird, sollte man noch einmal über Bedeutung und Nützlichkeit von Benchmarks nachdenken. Benchmarks dienen in erster Linie zum Abschätzen der Leistung von Programmen, die in höheren Sprachen auf Zielsystemen codiert sind, und, um ein Gefühl für die Leistung eines Systems bezüglich der Prozeduren zu gewinnen, z. B. die Benutzung des Editors, Compilers usw.

Die Systeme wurden zum größten Teil (90 %) vom Autor persönlich getestet. Hierbei stellte sich heraus, daß die Zeiten für die ICM-3216-Konfiguration und eine VAX etwa im gleichen Bereich lagen. Die Zeiten für die anderen Systeme waren um den Faktor 2...3 größer, in anderen Fällen sogar um den Faktor 10 oder 20. Dies ist außerordentlich interessant, weil das verwendete Plattenlaufwerk eine relativ lange mittlere Suchzeit hat (80 ms). Ein echter Test, der die Geschwindigkeiten von Compiler, Assembler und Linker erfordert das Erzeugen eines außerordentlich umfangreichen Programms. Dabei sollte man auch wirklich die Zeiten festhalten, die erforderlich sind, um in den Editor oder das Quellen-Code-Steuersystem zu gelangen oder diese zu verlassen. Der Autor konnte nicht alles testen, obwohl festzuhalten ist, daß bei den von ihm entwickelten und erzeugten Programmen der Editor außerordentlich schnell aufgerufen wurde und seine Aufgabe erfüllte.

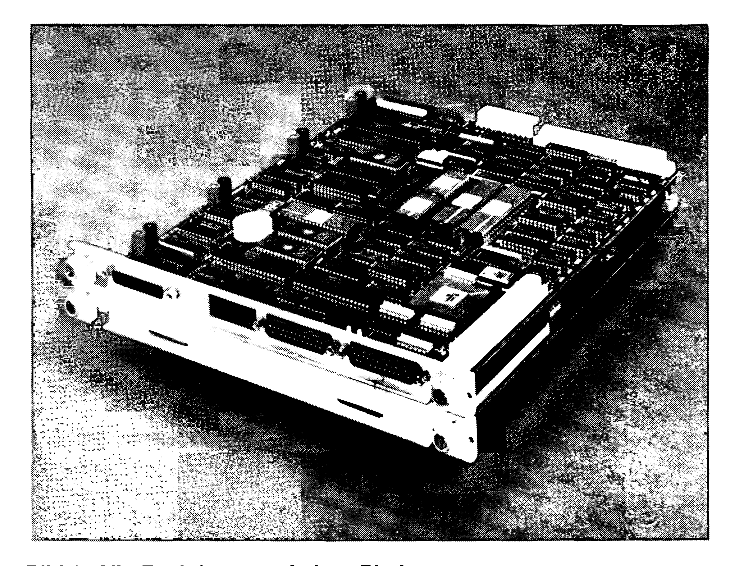


Bild 2. Alle Funktionen auf einer Platine: Der Boardcomputer ICM3216

Was die tatsächlichen Zeiten für das Abarbeiten von Programmen angeht, wird die Leistung eines jeden Algorithmus, der in einer höheren Sprache auf einer beliebigen Maschine codiert ist, von fünf Dingen bestimmt: die Effizienz des Compilers, die Geschwindigkeit der CPU, die Zugriffsgeschwindigkeit auf den Speicher, die Geschwindigkeit der E/A-Einheiten und die Implementierung, die der Programmierer gewählt hat. Die Geschwindigkeit der CPU hängt von deren Architektur und der Taktfrequenz ab. Die Zugriffsgeschwindigkeit auf den Speicher bestimmen mehrere Faktoren, aber insbesondere die Zugriffsgeschwindigkeit der Speicherchips und der Weg, über den sie mit der CPU verbunden sind. Neben dem Programmierer ist der Compiler vielleicht der größte Faktor für die Leistung, denn es ist ja schon fast Allgemeingut, daß die Optimierung von compilererzeugtem Code die Leistung um den Faktor 2...5 verbessert (Richtwerte).

Wenn also jemand die Geschwindigkeit für die Abarbeitung eines Programms beschleunigen will, ist es am wichtigsten, den Algorithmus zu überprüfen und als nächstes die Implementierung. Man erreicht beispielsweise grundsätzlich einen Verbesserungsfaktor von 2, wenn man dort, wo es geht, Registervariablen benutzt. Dies funktioniert bei allen Maschinen so lange, bis man mehr Variable hat als Register vorhanden sind. Daraus ergibt sich, daß manche Implementierungen schneller auf der einen Maschine als auf der anderen laufen, es sei denn jemand löst sein Implementierungsproblem mit weniger Registern. Die Geschwindigkeit der E/A-Einheiten ist nur für Programme interessant, die E/A-Funktionen ausführen. Dies ist sehr häufig der Fall, allerdings kann man die E/A-Geschwindigkeit eines Systems kaum verbessern. Außerdem kann kein Benchmark-Test Aussagen über die Leistungsfähigkeit eines Programms unter einer vorgegebenen Umgebungsbedingung machen.

Aufgrund der klaren Architektur und des orthogonalen Befehlssatzes (jede Instruktion arbeitet in jeder Adressierart) erfolgt die Umsetzung von Code auf höherer Sprachebene in Assemblersprache sehr schnell, so daß man erwarten kann, daß der C-Compiler brauchbaren Code erzeugt. Ein Überwachen des anfallenden Assembler-Sprachcodes durch den Compiler bestätigt dies, obwohl man diesen Code auch noch weiter optimieren kann und dadurch merkliche Verbesserungen erreicht. So führte beispielsweise die Verwendung von Registervariablen für die Array-Pointer im Sieve-Benchmark zu einer Halbierung der Laufzeit.

Wenn beim Testen von Systemen die Compiler etwa vergleichbaren Assembler-Sprachcode liefern, liegen die Leistungsunterschiede in erster Linie in E/A-, Speicher- oder CPU-Geschwindigkeit. Wenn eine Differenz

von 20% existiert, kann diese bereits merklich sein. *Tabelle 1* zeigt die Resultate für die Benchmarks, bei denen auf einigen Systemen Unix (oder ähnliches) läuft. Alle Zeiten sind in Sekunden angegeben. *Tabelle 2* zeigt als Bezug dazu die Resultate von Benchmarks für einige Systeme, die ohne Speicherverwaltung arbeiten.

Um alles in einen Zusammenhang zu stellen, sollen auch die Zeiten für eine VAX-11/780 mit Fließkomma-Hardware angegeben werden. In diesem Fall wurde das NOOP-Flag für den Compiler benutzt, weil sonst der Compiler den Benchmark-Test „weg-optimiert".

Tabelle 1. Benchmarkzeiten in s – für Systeme unter Unix

Prozessor	Takt	Betriebs-system	Fib	Sieve	Float
68010	10 MHz	Unisoft Unix	17,8	4,5	35,2
68000	10 MHz	–	51	13,6	277,4
80286	6 MHz	Xenix	30,8	6,85	12,8
80286	8 MHz	Xenix	17,8	3,7	8,7
32016	10 MHz	Unix	23,9	4,6	3,2

Das Symbol „–" anstelle von Daten zeigt an, daß hier keine Angaben vorlagen oder nicht gegeben werden konnten. In den meisten Fällen existierte keine Fließkomma-Unterstützungseinheit.

Interessant ist der weite Streubereich der Leistung von 68000-Systemen. Es scheint so, daß die Speicher-Zugriffsgeschwindigkeit einen wesentlichen Faktor zu diesen Differenzen beiträgt. In einem Fall, in dem es möglich war, einen Logik-Analysator zur Untersuchung der Hardware anzuschließen, stellte es sich heraus, daß die Zahl der Wartezyklen sechs oder mehr betrug. Im Gegensatz dazu benutzt das in den Tabellen erwähnte

Tabelle 2. Benchmarkzeiten in s für Syteme unter anderen Betriebssystemen

Prozessor	Takt	Benchmark	Fib	Sieve	Float
8088 (IBM PC)	4,77 MHz	Aztec C	82,2	18,5	118,5
8086 (+ 8087)	5 MHz	Aztec C	59,5	11,3	18,3
80186	8 MHz	Aztec C	30	7	45
68000 (VME)	12,5 MHz	–	32	7	75
Z80A	4 MHz	C80	130	33	–
6502 (Apple)	1 MHz	Aztec C	810	40	2700 löschbar
11/23+	–	Whitesmiths	53	10	30
68000 (Mac)	8 MHz	Aztec C	24,7	6,2	268,2
Zum Vergleich: VAX mit Fließkomma-Prozessor					
VAX 11/780–		DEC C-Compiler	24	3	4

68010-System einen eigenen Speichercontroller, was zur Folge hat, daß lediglich eine Platine benötigt wird, die in der Speicherverwaltungs-Operation mit einem Wartezustand auskommt.

Diese Angaben machen klar, daß es sich bei der Platine ICM3216 um ein sorgfältig entwickeltes System handelt, das nicht durch Wartezyklen verzögert wird. Tatsächlich war die ICM3216-Konfiguration in den Float-Benchmarks so gut, daß der Autor vom Compiler produzierte Assemblersprache näher untersuchte. Die Vermutung, daß der Compiler das Benchmark-Programm weg-optimiert, bewahrheitete sich nicht. Der Benchmark wurde in der geforderten Weise ausgeführt. Es stellte sich heraus, daß auf der Platine ICM3216 Befehle für doppelte Genauigkeit benutzt wurden.

Eine andere Frage in bezug auf die Leistung ist, ob das System auch in einer Mehr-Prozessor-Konfiguration läuft. Einige Versionen von Unix-ähnlichen Betriebssystemen haben bei mehreren Prozessen Probleme, weil die Kontext-Schalter eine merkliche Zeit beanspruchen. Weil das getestete System über 4 MByte Speicher verfügt und die Prozessoren der 32000-Serie eine solche klare Architektur haben, waren keine Probleme mit mehreren gleichzeitig laufenden Prozessen zu erwarten. Mit zwölf Dummy-Prozessen, die gleichzeitig ablaufen, betrug die zusätzliche Zeit für die oben genannten Tests etwa 1/10 s. Das gleichzeitige Abarbeiten mehrerer Tests verlängerte die Laufzeit noch weniger. Die notwendige Zeit war immer die Summe der erwarteten Laufzeiten für die seriellen Tests. Dies bedeutet eine gute Implementierung von Unix auf der Hardware.

Dies läßt das Preis-Leistungsverhältnis des ICM3216 deutlich werden. Es gibt noch einige andere Punkte, die vielleicht für potentielle Anwender dieser Produkte in Workstations interessant sein könnten. Erstens arbeitet man bei National Semiconductor an einem Platinenkonzept für den Prozessor 32032. Dies bedeutet eine zusätzliche Leistungssteigerung, bei der allerdings die vollständige Objekt-Code-Compatibilität erhalten bleibt. Als zweites existiert der „Minibus". Hierbei handelt es sich um einen 16-Bit-multiprozessor-orientierten synchronen Bus. Es ist nicht ganz klar, was National Semiconductor zukünftig mit dem Minibus plant, jedenfalls ist er für Entwickler von Multiprozessor-Systemen nicht uninteressant.

Man kann ihn anstelle einer Backplane verwenden, um zusätzliche intelligente E/A-Einheiten anzuschließen, wenn dies notwendig ist. Die Implementierung erfolgt mit Hilfe eines Chips, das leicht erhältlich und einfach zu benutzen ist. Als drittes wird demnächst ein leistungsfähiger Grafikbaustein auf den Markt kommen, der sich in Zusammenhang mit diesem Platinenkonzept bestimmt gut benutzen läßt. Damit ergeben sich sehr gute Möglichkeiten zur Produktentwicklung im CAD/ CAM-Bereich.

Jean-Claude Mathon

32-Bit-Architektur mit hoher Effizienz

Der Begriff „32-Bit-Architektur" ist in den vergangenen Jahren von den Mikroprozessor-Herstellern mehr oder weniger wahllos benutzt worden. Einige Mikroprozessoren werden fälschlicherweise als 32-Bit-Bausteine bezeichnet, weil sie einige Funktionen mit 32-Bit-Operanden ausführen können. Jedoch ist der interne Datenbus nur 16 Bit breit, und die Recheneinheit kann bei 32-Bit-Operationen nur nacheinander zwei 16-Bit-Operationen ausführen. Es ist deshalb wesentlich, wenn man über eine 32-Bit-Architektur spricht, ein 32-Bit-Register, eine 32-Bit-ALU und 32-Bit-breite interne Datenpfade zu meinen. Dies trifft auf die Prozessoren der Serie 32000 von National Semiconductor zu.

Langfristigkeit und Sicherheit zählen zusätzlich

Zu den in der Einleitung genannten Eigenschaften eines leistungsfähigen Mikroprozessors muß eine langlebige Architektur zum Schutz und Erhalt von Software-Investitionen hinzukommen.

Wenn der Hersteller von einer 32-Bit-Architektur spricht, ist folgendes gemeint:
- Effektive Unterstützung der Hochsprachen;
- Erzeugung eines kompakten Maschinencodes;
- Vollkommene Adaption der strukturierten Programmierung;
- Gleitkomma-Operationen;
- Virtuelle Speicherverwaltung nach dem „Demand-Paged"-Verfahren;
- Slave-Prozessor-Konzept.

Warum 32 Bit?

Die Entscheidung für 32-Bit-Prozessoren basiert im wesentlichen auf zwei elementaren Faktoren:
- Architektur und/oder
- Busbreite.

Zunächst die Begründung dafür, wieso man für eine gute Architektur 32-Bit-Technik braucht:

Die 32-Bit-Technik ist für eine effektive Unterstützung der Hochsprachen eine Notwendigkeit. Zum Beispiel sind die Adressenzeiger, die häufig von Hochsprachen manipuliert werden, 32 Bit breit, um auf einen genügend großen Speicherraum zugreifen zu können. Die bearbeiteten Zahlen sind entweder Real- oder Integerzahlen. Integerzahlen mit 32 Bit erlauben eine Darstellung von 10stelligen Dezimalzahlen, 16 Bit gestatten nur fünf Stellen, eine ungenügende Genauigkeit für Standardrechnungen. Analoge Daten werden normalerweise mit 12 oder 16 Bit digitalisiert, ihre Verarbeitung aber, z. B. zur Filterung oder Prozeßsteuerung, schließt Multiplikationen und Additionen ein, deren Zwischenergebnisse im allgemeinen mit 32 Bit codiert werden müssen. Auch die Gleitkomma-Darstellung für Realzahlen ist mit 32 Bit standardisiert (IEEE, Single-Precisson-Format). Um diese Daten effektiv zu bearbeiten und um zu vermeiden, daß man solche Operationen in mehrere Arbeitsschritte zerlegen muß, ist eine 32-Bit-Architektur notwendig.

Sie ermöglicht auch, einen linearen Speicher von 4 GByte zu adressieren und ihn einfach und direkt zu bearbeiten, z. B. zur schnellen Verwaltung von großen Datenbanken.

32 Bit ist die ideale Breite, um einem Produkt gut zehn Jahre Lebenszeit zu garantieren. Man darf nicht vergessen, daß die Software der weitaus kostspieligste Faktor bei der Entwicklung einer Mikroprozessor-Anwendung ist. Da der Umfang der Programme nach wie vor ständig zunimmt, ist es für den Anwender lebenswichtig, den Wert seiner Investitionen in die Software-Entwicklung zu erhalten sowie die Sicherheit zu haben, daß die Architektur für zukünftige Produkte der Familie tragfähig ist und daß die Kompatibilität erhalten bleibt.

CPU-Leistung steigt mit

Die Busbreite beeinflußt natürlich auch direkt die Leistungsfähigkeit der CPU. Eine CPU mit einem 32 Bit breiten Bus erfaßt bei einem Speicherzugriff gleichzeitig 32 Bit, was alle Prozesse beschleunigt, die sehr speicher-

intensiv sind. Bezogen auf die Systemumgebung eines 32-Bit-Mikroprozessors war es eines der Ziele, den Systembus weniger häufig durch Speicherzugriffe oder E/A-Operationen der CPU zu belegen, so daß der Bus wesentlich mehr für DMA-Verkehr, Übertragungen zu einer grafischen Einheit oder für weitere Prozessoren zur Verfügung steht. Im Mittel benutzt die CPU NS32032 nur etwa 40 % der zur Verfügung stehenden Bandbreite des Systembusses. Aus diesem Grund hat man z. B. die CPU NS32132 entwerfen können, die mit dem Typ NS32032 identisch ist, aber zusätzlich die Überwachungslogik an Bord hat, um mit zwei Prozessoren ohne weitere externe Logik parallel auf demselben Systembus arbeiten zu können.

Die Systemlösung

Mit der Serie 32000 wurde erstmals eine Mikroprozessor-Familie entworfen, bei der man zunächst analysierte, welche Software später darauf laufen sollte. Beim Entwurf dieser Familie gab es das klare Ziel: die Software-Entwicklung sollte so effektiv und produktiv wie möglich sein. Die Serie 32000 unterstützt deshalb Hochsprachen sehr umfangreich und erzeugt gleichzeitig einen äußerst kompakten Code. Dies gibt dem Anwender die Möglichkeit, fast alle seine Programme in Hochsprachen wie Pascal, C oder Fortran zu schreiben.

Es ergeben sich daraus wichtige Vorteile:
- Kürzere Zeiten zum Programmieren, Fehlersuchen und Testen;
- Höhere Zuverlässigkeit der Programme;
- Leichte Programmpflege;
- Einfache Dokumentation;
- Optimale Anlage von Investitionen in die Software;
- Zum Teil prozessorunabhängige Anwender-Software;
- Über einen langen Zeitraum gesicherte Software.

Deshalb sollte ein Mikroprozessor nicht nur mit dem Blick auf die CPU alleine beurteilt werden. Heute muß ein 32-Bit-Mikroprozessor nach den gleichen Maßstäben beurteilt werden, wie bislang ein OEM-Minicomputer. Daher waren die wesentlichen Gesichtspunkte beim Entwurf der Serie 32000:
- Systemleistung in einer typischen 32-Bit-Umgebung;
- Effektivität des Entwurfes und der Ausführung der Software;
- Frühe Verfügbarkeit einer kompletten Lösung und aller notwendigen Hilfsmittel und Werkzeuge.

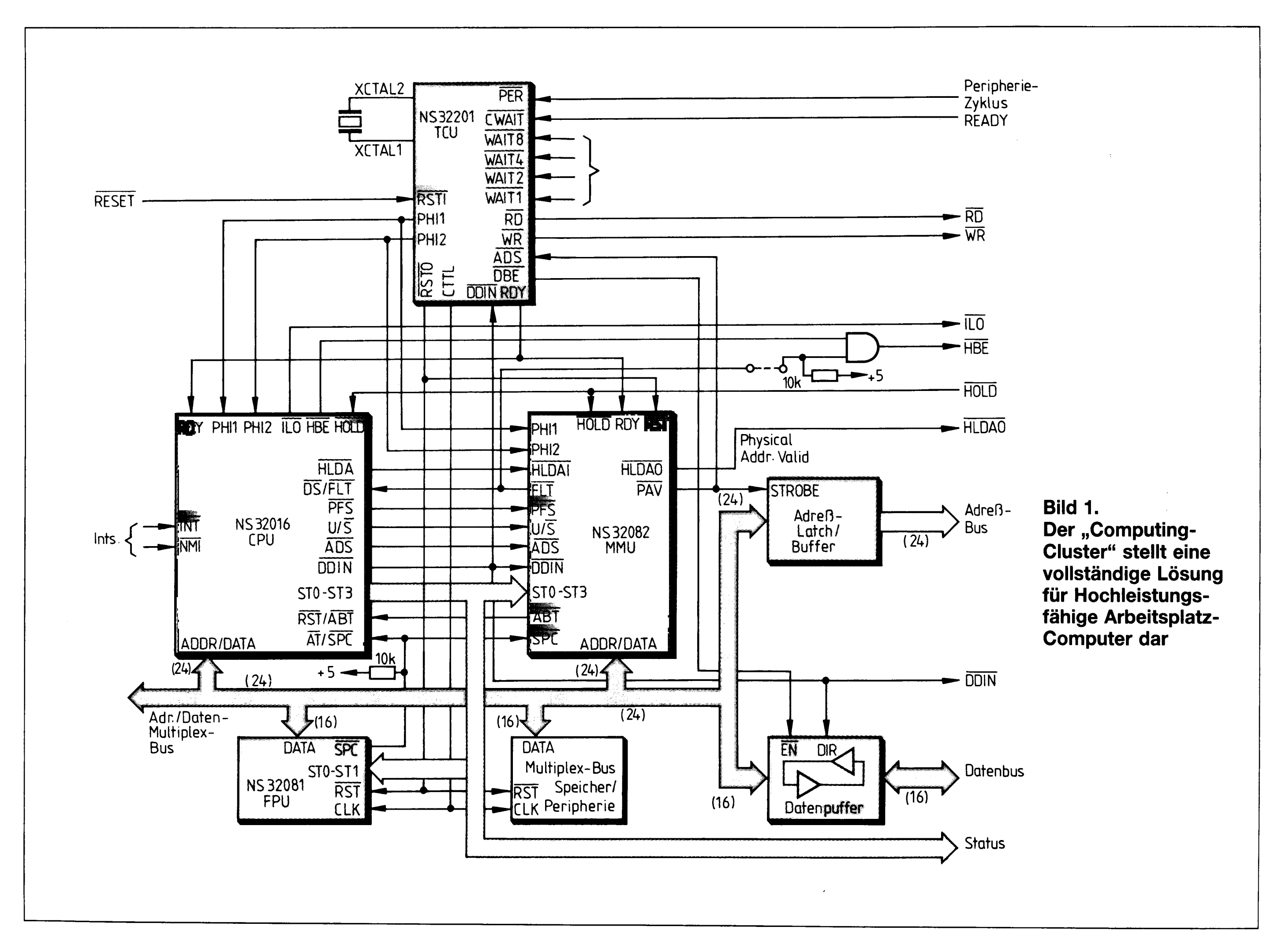


Bild 1. Der „Computing-Cluster“ stellt eine vollständige Lösung für Hochleistungsfähige Arbeitsplatz-Computer dar

Das erlaubt dem Anwender sehr schnell, mit seinem neuen Produkt auf den Markt zu kommen. Es ist für den Anwender wichtig, daß ein kompletter Satz von Bausteinen (CPUs, MMU, FPU, ICU, TCU, DMA usw.) in Produktions-Stückzahlen zur Verfügung steht, damit er sich voll auf seine Aufgabenstellung konzentrieren kann.

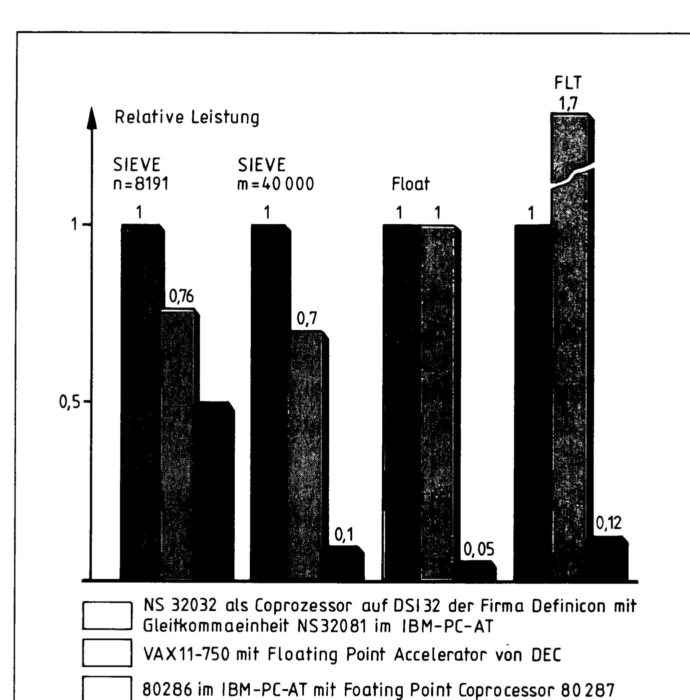


Bild 2. NS32032 als Coprozessor auf DSI32 der Firma Definicon mit Gleitkomma-Einheit NS32081 im IBM-PC-AT VAX11-750 mit DEC-Floating-Point-Accelerator 80286 im IBM-PC-AT mit Floating-Point-Coprozessor 80287 Benchmark-Ergebnisse, wie sie im August 1985 vom „Byte"-Magazin veröffentlicht wurden:
- **Der Sieve-Benchmark gibt darüber Auskunft, wie gut Boolesche Algebra und Berechnung von Integerzahlen, in Hochsprachen implementiert, ablauffähig sind.**
- **Der Float-Benchmark testet die Leistungsfähigkeit, Gleitkomma-Operationen auszuführen.**
- **Der FLT-Benchmark testet die Geschwindigkeit des Gleitkomma-Prozessors.**

Konzept für eine ganze Familie

Die Zentraleinheiten NS32032 (externer 32-Bit-Bus), NS32016 (externer 16-Bit-Bus) und NS32008 (externer 8-Bit-Bus) haben alle die gleiche interne Architektur (interner 32-Bit-Bus, 32-Bit-Register, 32-Bit-Recheneinheit), den gleichen voll symmetrischen Befehlssatz, die gleichen Adressierungsarten, die Hochsprachenunterstützung, und alle Prozessoren verarbeiten die gleiche Software. Es gibt somit eine echte Auf- und Abwärtskompatibilität zwischen allen Prozessoren. Es wurde die gleiche, einfache, klare und leistungsfähige Architektur in Prozessoren mit 8-, 16- und 32-Bit-breiten externen Bussen implementiert, um die unterschiedlichen Leistungsanforderungen, abhängig von der Anwendung, zu unterstützen. Zusätzlich stehen Slave-Prozessoren (FPU, MMU) und weitere periphere Bausteine wie intelligente Interrupt-Controller (ICU), DMA, UART, Dynamic-RAM-, LAN-, Hard- und Floppy-Disk-Controller, Terminal-Management-Prozessoren (TMP) und leistungsfähige Gate-Arrays zur Verfügung, die eine Lösung der Probleme auf Systemebene ermöglichen.

Eine klare, symmetrische Architektur

Die klare Architektur der Serie 32000 wird oft mit der Architektur der Minicomputer VAX-11 verglichen. Das heißt, daß ein Ingenieur, der Programmier-Erfahrungen mit der VAX hat, ein bekanntes Umfeld vorfindet, wenn er beginnt, Programme für Anwendungen der Serie 32000 zu entwickeln. Wenn man ein sehr leistungsfähiges System, basierend auf der CPU NS32032 bauen will, ist der Vergleich zur VAX sicher interessant. Der Anwender der klassischen 8- und 16-Bit-Mikroprozessoren aber könnte davon abgeschreckt werden. Er wird vielleicht meinen, daß er keine 32-Bit-Architektur braucht, weil diese zu teuer und für seine Anwendung nicht verfügbar sei.

Wie soll man darüber denken? Eine 32-Bit-Architektur mag zunächst vielleicht erschreckend sein, aber schauen wir uns doch einmal die Architektur eines 8-Bit-Mikroprozessors an: Der Befehlssatz macht es hier erforderlich, daß immer einer der Operanden in einem Register oder dem Akkumulator sein muß. Die jeweils gültigen Adressierungsarten sind für jeden Befehl gesondert spezifiziert. Der Befehlssatz ist nicht symmetrisch. Die Möglichkeiten, in Hochsprachen zu programmieren, sind beschränkt, weil unakzeptabel lange und langsame Programme entstehen. Es wird keine oder nur wenig Unterstützung für Gleitkomma-Operationen geboten, und es ist schwierig, 16-Bit-Daten und 32-Bit-Zeiger zu verarbeiten. Die klassische Architektur der 8- oder 16-Bit-Mikroprozessoren ist nicht gerade anwenderfreundlich! Im allgemeinen bedeutet dies, daß die Arbeit eines Programmierers durch die vielen Einschränkungen und Grenzen nicht komfortabel ist, sowohl im Assembler- als auch im Hochsprachen-Bereich.

Im Gegensatz dazu haben alle Prozessoren der Serie 32000 die gleiche 32-Bit-Architektur, und es steht der gleiche völlig symmetrische Zwei-Operanden-Befehlssatz zur Verfügung. Jeder Operand kann in einem Register oder irgendeiner Speicherstelle stehen und durch alle verfügbaren Adressierungsarten angesprochen werden. Jeder Befehl kann ohne Unterschied Bytes, Worte, Doppelworte oder auch Gleitkomma-Zahlen bearbeiten, soweit es sinnvoll ist. Das erleichtert die Arbeit eines Programmentwicklers im Assemblerbereich erheblich, macht Hochsprachen-Compiler wesentlich effektiver bezüglich der Codelänge und reduziert die Ausführungszeiten merklich.

Des weiteren unterstützen alle Prozessoren der Serie 32000 Hochsprachen gleich wirkungsvoll. In der Tat sind nicht nur typische Hochsprachen-Operationen direkt im Maschinencode implementiert, sondern auch typische Datenstrukturen, die dabei verarbeitet werden. Dies wird vor allem erreicht durch:
- Allgemeine Register;
- Leistungsstarke und an Compiler angepaßte Adressierungsarten;
- Spezielle Befehle, die Bit- und Bitfeld-Operationen erlauben (Bearbeitung von „Packed-Arrays" und Records);
- Leichte Handhabung von mehrdimensionalen Datenfeldern, unabhängig davon, ob es sich um Byte-, Wort- oder Doppelwort-Elemente handelt;
- Komplexe Behandlungsmöglichkeiten für Zeichenfolgen (Character-Strings);
- Direkte Implementierung des CASE-Befehls von Pascal (mehrfache Sprungverzweigung, abhängig vom Wert eines Parameters);
- Handhabung von Semaphoren für Echtzeit-Anwendungen.

Bei der 32000-Familie ist sichergestellt, daß der Gleitkomma-Slave-Prozessor (FPU, Floating-Point-Unit) an jede der CPUs angeschlossen werden kann, wenn höhere Genauigkeit für die Berechnung notwendig ist. Die FPU von National Semiconductor verarbeitet Gleitkomma-Zahlen mit einfacher (32 Bit) und doppelter (64 Bit) Genauigkeit nach dem IEEE-Standardformat. Die FPU ist ein Slave-Prozessor, das heißt, Gleitkomma-Befehle werden von der CPU decodiert und unter ihrer Kontrolle ausgeführt. Die Datenübertragung zwischen der CPU und der FPU geschieht automatisch mit einem einfachen Protokoll, das die schnelle Übertragung von 16-Bit-Informationsblöcken in zwei Taktzyklen (auch für die CPU NS32008) erlaubt, ohne zusätzliche Befehle im Programmcode zu benötigen. Für den Benutzer sieht es so aus, als sei die FPU Teil der CPU. Die FPU enthält weitere acht allgemeine 32-Bit-Register, die von der CPU genauso wie ihre eigenen internen Register angesprochen werden können.

32-Bit-Anwendungen

Der 32-Bit-Markt befindet sich derzeit in seiner Anfangsphase, und National Semiconductor kann ein vollständiges Produktspektrum einschließlich CPUs, MMU, FPU, TCU und ICU bieten. Die Hauptbereiche, in denen heute 32-Bit-Mikroprozessoren Anwendung finden, sind:

- CAD/CAE-Arbeitsplätze;
- Büro-Arbeitsplätze oder Büroautomatisierungs-Computer;
- Bildverarbeitung oder Bildsynthetisierung;
- Datenbank-Verwaltungen;
- Automatische Testsysteme;
- Ausfallsichere Rechner;

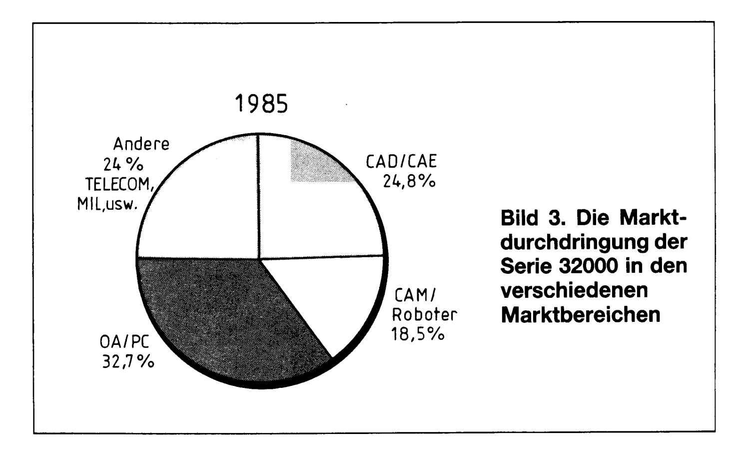


Bild 3. Die Marktdurchdringung der Serie 32000 in den verschiedenen Marktbereichen

- Roboter- und Prozeßsteuerungen;
- CAM-Anwendungen;
- Datenerfassungs- und Telekommunikations-Anlagen.

Ein großer Teil der 32-Bit-Anwendungen bezieht sich auf Unix. Es stehen alle wichtigen Unix-Versionen zur Verfügung:

- Genix-4.1 und Genix-4.2, abgeleitet von Unix-Berkeley-4.1 und -4.2;
- Unix-System-V, Release-2.0, Version 2, von AT&T mit „Demand-Paged" und File-/Record-Locking;
- Xenix-32 von Microsoft.

Diese vier Versionen unterstützen alle die für Mikroprozessoren neuartige virtuelle Speicherverwaltung nach dem „Demand-Paged"-Verfahren.

Die 32000-Familie in 8- und 16-Bit-Anwendungen

Bezogen auf den 8-Bit-Markt besteht bei National Semiconductor nicht die Absicht, mit Produkten wie dem Z80 oder ähnlichen Prozessoren zu konkurrieren. Die Zielrichtung sind anspruchsvolle 8-Bit-Anwendungen, bei denen die Leistungsfähigkeit traditioneller 8-Bit-Mikroprozessoren begrenzt ist und der Anwender nicht den Schritt zu einem 16 Bit breiten Bus machen möchte. Erfahrungen der Anwender, die NS32016 und MC68010 vergleichen, zeigen, daß mit dem 32016 die Programme nur halb so lang werden und ca. 40 Prozent schneller ablaufen. Dies gilt für Programme ohne Gleitkomma-Operationen. Bei Gleitkomma-Operationen werden die Unterschiede noch gravierender. Ein ähnliches Verhältnis ergibt sich aus dem Vergleich von 32008 und 68008.

Um den 8- und 16-Bit-Markt wirkungsvoll zu bedienen, gibt es Starter-Kits, umfangreiche Pakete mit Entwicklungs-Software, ein kostengünstiges Entwicklungssystem, einen Echtzeit-Betriebssystem-Kern, einen anspruchsvollen Baustein zur Unterbrechungssteuerung (ICU), CMOS-Versionen, Multibus-Platinen und eine neue 32-Bit-Architektur zum Preis einer „alten" 16-Bit-Architektur.

Umfeld der Serie 32000

Es ist für den 8- und 16-Bit-Anwender notwendig, daß er mit Hilfsmitteln versorgt wird, die es ihm leicht und kostengünstig ermöglichen, die Eigenschaften der Serie 32000 selber zu beurteilen. Das Starter-Kit (siehe Beschreibung an anderer Stelle dieses Heftes) gibt jedem die Möglichkeit, auszuprobieren, was er von 32000-Mikroprozessoren und den peripheren Bausteinen in seiner Systemumgebung erwarten kann.

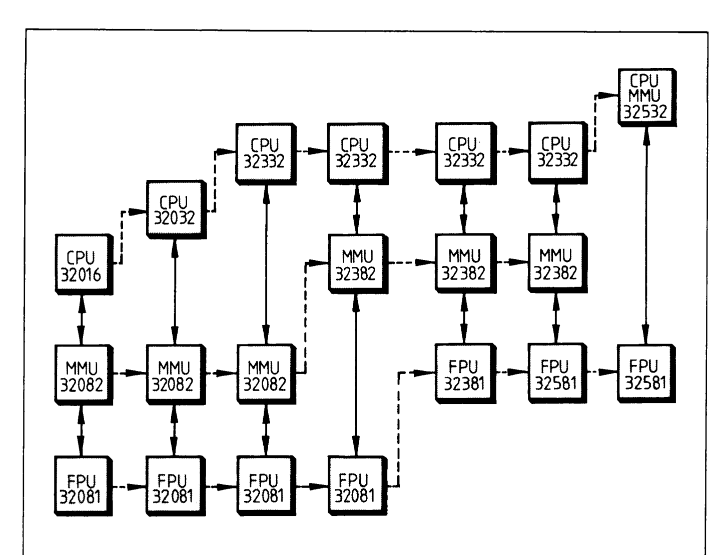


Bild 4. Ein klarer Entwicklungsweg in die Zukunft: zu jedem Zeitpunkt der Entwicklung steht ein vollständiger Computing-Cluster zur Verfügung

Wenn der Anwender die 32000-Familie einsetzen will, werden ihm mehrere Möglichkeiten angeboten. Hat er bereits Rechner im Hause, wie eine VAX-11, eine PDP-11, einen IBM-PC, ein MDS oder eine CP/M-Z80-Maschine, stehen alle erforderlichen Cross-Software-Pakete zur Verfügung einschließlich Assembler, C- und/oder Pascal-Compiler, Link-Editor, Bibliotheken und symbolische Debugger. Preis und Leistungsfähigkeit dieser Werkzeuge hängen natürlich von der Wahl des verwendeten Rechners ab.

Sind diese Rechner allerdings schon durch andere Aufgaben überlastet oder schwer verfügbar oder besteht die Notwendigkeit, ein Entwicklungssystem zu haben, das auf der Serie 32000 basiert, um Software zu testen oder dieses System als Zielsystem zu verwenden, so werden zwei Alternativen angeboten:

Einmal das SYS32, basierend auf dem kompletten Serien-32000-Chipsatz (10 MHz), als Mehrplatz-Umgebung für acht Benutzer. Es hat eine Winchesterplatte von 70 MByte (erweiterbar bis 490 MByte), eine 20-MByte-Streamer-Bandeinheit für Backups und 1,25 MByte Arbeitsspeicher (erweiterbar bis 3,25 MByte). Das Betriebssystem ist Genix-4.1 (Berkeley-4.1-bsd) mit umfangreichen Erweiterungen zur Unterstützung der Software-Entwicklung. Es stehen natürlich alle Entwicklungs-Hilfsmittel, wie bei den Cross-Software-Paketen, zur Verfügung.

Die zweite Lösung ist das System VR32 für zwei Benutzer, das als Entwicklungssystem und als Zielsystem verwendbar ist, da es hardware- und softwaremäßig konfiguriert werden kann. Es basiert ebenso auf dem kompletten 32000-Chipsatz (10 MHz) und ist auf Multibus-Platinen aufgebaut. VR32 wird mit Unix-System-V, Release 2.0, Version 2 sowie einem konfigurierbaren E/A-System ausgeliefert und enthält alle Entwicklungshilfsmittel, die zusammen mit dem SYS32 geliefert werden.

Für die Serie 32000 gibt es auch einen Echtzeit-Betriebssystem-Kern namens „Exec“, dessen Quellcode zu einem attraktiven Preis vertrieben wird und der dem Anwender völlig frei zur Verfügung steht. Der Anwender kann um diesen Kern herum entsprechend seinen Anforderungen ein Echtzeit-System und anschließend seine Programme bauen. Für das endgültige Produkt muß der Kunde keine Lizenzgebühr zahlen. Exec hat Schnittstellen zu Pascal und C. Das ermöglicht dem Anwender, Programme in Hochsprachen zu schreiben.

Im folgenden Teil geht es um die verschiedenen Applikations-Marktsegmente und um Anwendungen der Serie 32000, soweit diese veröffentlicht werden dürfen.

CAD-Arbeitsplatzrechner:

NS32032/NS32016 und Unix/Berkeley-4.2

Bei der Analyse des 32-Bit-Marktes fällt auf, daß er zunächst der aktiveste Bereich im Arbeitsplatz-Rechner- und CAD-Markt war. In diesem Bereich dominierten für einige Jahre die VAX-Computer. Kleinere Maschinen basierten auf 16-Bit-Mikroprozessoren. Solche Systeme, gebaut für den Ingenieur, mußten als wichtiges Leistungsmerkmal Gleitkomma-Operationen ermöglichen. Darüber hinaus setzten die meisten Anwender der VAX das Betriebssystem Unix/Berkeley-4.1-bsd ein.

Es ist nicht überraschend, daß heute die Serie 32000 für die meisten Arbeitsplatz-Rechner in diesem Bereich die Basis bildet. Die Gleitkomma-Einheit (FPU) NS32081 bietet die für diese Anwendung erforderliche Rechenleistung. „Genix“, abgeleitet von Unix/Berkeley-4.1, für die Serie 32000 ist leistungsfähiger als die VAX-Implementierung. Diese Überlegenheit basiert auf verschiedenen Funktionen, wie z. B. der Verwaltung des Referenzbits in der MMU der Serie 32000.

Die Serie 32000 wurde bislang u. a. von folgenden OEM-Anwendern gewählt:

- Intergraph (Professional-Workstation Interpro 32, Interact 32, basierend auf NS32032-CPU, MMU, FPU sowie Berkeley-4.2 und System-V);
- Mosaic Technologies;

- Encore Computer Corporation (Arbeitsplatz-Computer, basierend auf CPU NS32032);
- Tektronix (Grafik-Arbeitsplätze der Serie 6100 mit CPU NS32016);
- Atari (Workstation, basierend auf CPU NS32032);
- Daisy (Workstation, basierend auf CPU NS32032);
- Saber Technologie (Hochleistungs-Workstation, basierend auf zwei CPUs NS32032);
- Whitechapel-Computer-Works (Personal-Workstation MG-1, basierend auf NS32016 und Berkeley-4.2);
- Diab (DS60, grafischer Arbeitsplatz, basierend auf NS32032 und D-NIX-Betriebssystem, Unix-ähnlich mit Echtzeit-Eigenschaften);
- Telesis-Systems (Workstation mit NS32032 und Unix-System-V-Coprozessor);
- und vielen anderen.

Büro-Computer:

NS32032/NS32016 mit Unix-System-V oder Xenix-32

Der Marktbereich der Büroautomatisierung folgte den Spuren der Arbeitsplatz-Computer. In diesem Marktsegment wird die Nachfrage nach Unix ständig größer, und der Trend bewegt sich eindeutig auf eine Standardisierung hin. Die Unix-Versionen, die hier benötigt werden, sind nicht von Unix-Berkeley abgeleitet, sondern sind Unix-System-V (Standard von AT&T) oder Xenix, das von Microsoft angeboten wird. Die Anwenderprogramme werden mehr und mehr konsequent durchstrukturiert und benötigen zunehmend Platz im Arbeitsspeicher, um wirkungsvoll und schnell ablauffähig zu sein. Im allgemeinen handelt es sich um Mehrplatz- und Multitasking-Systeme. Nur die virtuelle Speicherverwaltung nach dem „Demand-Paged"-Verfahren erlaubt mehreren Benutzern, große Programme mit akzeptablen Reaktionszeiten simultan zu benutzen und dabei mit einem von den Kosten her vertretbaren großen Arbeitsspeicher (RAM) auszukommen. Die virtuelle Speicherverwaltung nach dem „Demand-Paged"-Verfahren vermeidet, daß das System durch zu häufige Datenübertragungen zwischen Platten- und Arbeitsspeicher überlastet wird, wie es bei der virtuellen Speicherverwaltung nach dem segmentierten Verfahren der Fall ist.

Auch für diesen Bereich des Marktes ist die Serie 32000 interessant. Unix-System-V/Serie 32000, Release 2.0, Version 2, validiert von AT&T, und Xenix-32 sind seit längerem verfügbar. Es sind die einzigen Unix-Versionen, die im Bereich der kommerziellen Anwendungen das „Demand-Paged"-Verfahren unterstützen. Und um die Wichtigkeit dieses Arguments noch zu unterstreichen, sei daran erinnert, daß alle großen Minicomputer-Hersteller (IBM, DEC, Data General usw.) sich schon seit einigen Jahren für das „Demand-Paged"-Verfahren bei der virtuellen Speicherverwaltung entschieden haben.

Die Serie 32000 wurde im Bereich der Büro-Automatisierung gewählt von:

- Burroughs Corporation (NS32032 und System-V);
- Nixdorf (NS32016 und Nicos, ein Unix-ähnliches Betriebssystem);
- Prime Computer;
- Siemens (PC-MX2-Mehrplatzsystem, NS32016 und Sinix);
- Datapoint;
- Diab (DS90, Mehrplatz-Supermikro, basierend auf NS32032 und D-NIX);
- Symetric-Computer-Systems (Symetric-375, 32-Bit-Arbeitsplatz-Computer);
- ICL, Dänemark (neue Generation von Büro-Computern, basierend auf NS32016 und Genix);
- und anderen Herstellern.

Häufig wird die CPU NS32016 auch als E/A-Prozessor oder in Gateway-Anwendungen für Großrechner und Minicomputer eingesetzt, wie z. B. von National Advanced Systems.

	32 Bit Mai 1984 %	16 Bit oder mehr April 1983 %	16 Bit oder mehr Mai 1982 %
Motorola	67,9 (1)	74,8 (1)	66,3 (1)
National Semi Corp.	47,6 (2)	15,6 (4)	5,4 (5)
Intel	46,0 (3)	70,9 (2)	65,2 (2)
Zilog	10,3 (4)	33,0 (3)	34,8 (3)
AMD	3,2 (5)	6,4	5,4 (5)
AT&T	2,8	—	—
Hewlett-Packard	2,4	—	—
Digital Equipment	2,0	2,5	4,3
Texas Instruments	2,0	11,7 (5)	20,7 (4)
Hitachi	1,6	1,8	2,2
NEC	1,6	2,5	3,3
NCR	1,2	—	—
Rockwell	1,2	1,4	3,3
Signetics	1,2	—	—

Bild 5. Zusammenstellung der Untersuchungsergebnisse von „Electronic Design": Der Trend für National Semiconductor ist klar

CAM und Robotersteuerungen:

Große Leistungsfähigkeit notwendig

Der Marktbereich CAM und Robotertechnik befindet sich heute in einer starken Entwicklung. Das vorrangige Bedürfnis hier ist generell große Computerleistung, die Fähigkeit, schnell und genau Bewegungen der verschiedenen Roboterachsen berechnen zu können. Weiter ist die Leistungsfähigkeit des Gesamtsystems wichtig und schließlich die Konfigurierbarkeit des endgültigen Systems, um es an die speziellen Erfordernisse anzupassen. Die Serie 32000 bietet dank der FPU die entsprechende Rechenleistung und gibt durch die Hochsprachen-Unterstützung die Möglichkeit, Systeme leicht zu konfigurieren und gleichzeitig ein sehr hohes Leistungsniveau zu erzielen. Unter anderem wurde die Serie 32000 von Bosch, General Robotics und Xydac Inc. (früher Modular) ausgewählt.

In hierarchischen Systemen, wie z. B. Anwendungen im Prozeßsteuerungs- und -überwachungsbereich, kann als zentrales System eine Einheit mit NS32032-CPU, MMU, FPU und Genix eingesetzt werden, die ihrerseits mit verschiedenen Datenkonzentratoren, basierend auf der NS32016-CPU, FPU, ICU und Exec (Echtzeit-Kern), verbunden ist. Diese Datenkonzentratoren sind mit den einzelnen Datenaufnahme-Modulen verbunden, die auf der NS32008, dem Interrupt-Controller ICU und Exec aufgebaut sind. Mit einem solchen Konzept wird vermieden, daß sich Wissen und Erfahrungen auf verschiedene Mikroprozessor-Familien aufteilen. Die einzelnen Arbeitsgruppen, die in den verschiedenen Ebenen des Projektes arbeiten, können einfach miteinander kommunizieren und ihre Erfahrungen austauschen. Eine erhebliche Einsparung der aufzuwendenden Entwicklungszeit und Investitionen ergibt sich durch eine einheitliche Sprache, gemeinsame Entwicklungswerkzeuge, eine Architektur und eine Systemphilosophie. Der Einsatz von Investitionen kann so optimiert werden.

Mehrprozessor-Systeme:

Modulare Leistung oder ausfallsicher

In Mehrprozessor-Systemen werden mehr und mehr leistungsfähige Mikroprozessoren eingesetzt. Dabei gibt es im allgemeinen zwei Zielrichtungen: Modulare Rechner, bei denen sich die Rechnerleistung schnell und unkompliziert an die sich verändernden Bedürfnisse der Anwender anpassen läßt, und ausfallsichere Rechner, die immer einwandfrei arbeiten und Fehler selbst erkennen müssen. Die modulare Leistungsfähigkeit ist für Anwendungen wie Bildverarbeitung und -synthese, automatisches Testen und generelle Computeranwendungen interessant.

Ausfallsichere Systeme werden vor allem in Bereichen wie Telefonvermittlung, Flugüberwachung und Flugreservierung benutzt. Für die Wahl der 32000-Familie gelten hier die gleichen Argumente wie für die modularen Systeme. Beispiele sind die Firmen Tolerant Computer Systems und Sequent Computer, die einen Rechner unter Unix auf der Basis von 30 CPU-Bausteinen 32032 anbieten, der eine Leistung von 21 Mips hat.

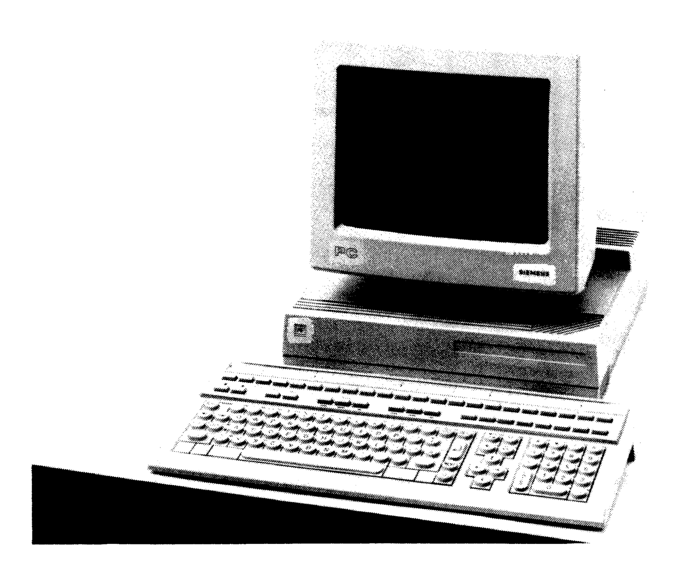
Allmählich gewinnt die Serie 32000 im Bereich der OEM-Systeme an Bedeutung, da es dem Anwender möglich ist, die Leistung einer VAX auf einer Platine mit einem erheblich günstigeren Preis zu integrieren. Der Integrationsgrad und die frühe Verfügbarkeit dieser Familie sind die wesentlichen Gründe hierfür. Beispiele:

- Goodspeed (GS-32-Einplatinen-Computer mit NS32032, MMU, FPU, ICU und VME, Multibus-I- oder -II-Schnittstelle);
- Compupro (VME-Boards);
- Elite-Computer (VME-Boards);
- Sritec (IBM-PC-Add-in);
- Opus Systems (IBM-PC-Add-in mit Unix-System-V);
- Definicon Systems Inc. (IBM-PC-Coprozessor-Platine mit NS32032, die die Leistung des PC-AT in den Schatten stellt);
- ICL Dänemark;
- HighTec (IBM-PC- und Intel-MDS-Add-on, außerdem ein Einplatinen-Unix-System);
- Spacelabs Computer;
- Argonne Systems;
- Janz (VME-Boards, Single-Boards);
- Gespac (G64-Boards);
- Elektroniklabor Hoffmann (ECB-, Multibus-II-Platinen);
- Benchmark Technology (Einplatinen-Grafikrechner).

Schutz der Software-Investitionen

Die einzelnen Anteile der Marktbereiche, in denen heute die Serie 32000 Eingang gefunden hat, spiegeln exakt die Untersuchungsergebnisse für den 32-Bit-Markt von Dataquest wider. Ungefähr 60 % des Marktes werden von Arbeitsplatz-Computern wie CAD/CAE- und Büro-Automatisierungssystemen eingenommen. Dies waren die ersten Bereiche, in denen eine starke Nachfrage nach 32-Bit-CPUs durch die großen Leistungsanforderungen und die Konkurrenzsituation entstanden ist. Die meisten Anwender haben sich hier, wegen der Leistungsfähigkeit und Vollständigkeit, für die Serie 32000 entschieden. Außerdem spielte die damit verbundene erhebliche Zeitersparnis, ein neues Produkt auf den Markt zu bringen, eine große Rolle.

Die Erhaltung des Wertes der Kunden-Software ist die vorrangige Zielsetzung von National Semiconductor. Zentrale Strategie ist: keine Änderung der Architektur (der Software-Schnittstelle), aber ständige Verbesserung der Umsetzung in Silizium, so schnell wie die Technologie es erlaubt.



Sinix-PCs mit 32000-Chipsatz

Wenn ein Systemhersteller sich für Unix oder eines der Unix-Derivate entscheidet, so ist damit in technischer und ökonomischer Hinsicht eine langfristige Festlegung verbunden. Das bezieht sich in nicht minderem Maße auf das Herzstück eines Computers: die CPU mit dem dazugehörigen Chipsatz. Die Siemens AG wählte nach sorgfältigen Erwägungen aller technischen und wirtschaftlichen Merkmale unter verschiedenen Prozessorfamilien die Serie 32000 von National Semiconductor für ihre neuen Sinix-PCs (MX2, MX4 und PC-2000). Zur Anwendung kommt die CPU 32016, die intern mit einem 32-Bit- und extern mit einem 16-Bit-Bus arbeitet, die zugehörige Speicherverwaltungs-Einheit (MMU) und die Fließkomma-Einheit (FPU). Der folgende Beitrag gibt Aufschluß über diese 32016-Applikation und die Leistungsmerkmale der darauf basierenden Sinix-Systeme.

Ein Mehrbenutzer-System ist anders...

...als eine Ansammlung von PCs. Obwohl es allgemein bekannt ist, daß ein echtes Mehrbenutzer-System einer entsprechenden Anzahl von Einzel-PCs in vielerlei Hinsicht überlegen ist, gab es mit Multiuser-Systemen in der Praxis bislang zwei Probleme: einerseits betraf das den Mangel an einem adäquaten Datenschutzmechanismus, der bei Zugriff mehrerer Benutzer auf dieselben Daten notwendig ist, andererseits stiegen die Antwortzeiten für die Benutzer stark an, wenn der Hauptprozessor ausgelastet oder überlastet war.

Beide Probleme konnten in den neuen Sinix-MX-Computern weitgehend gelöst werden. Leistungsprobleme sind durch die CPUs 32016 aus der Welt geschafft, weil diese Prozessoren mit ihrer internen 32-Bit-Busstruktur, einer neuartigen Architektur und dem zugehörigen Chipsatz (MMU, FPU) bei einer Taktfrequenz von 10 MHz ausreichende Leistungsreserven auch im Mehrbenutzer-Betrieb bieten. Ein wesentlicher Faktor ist dabei die von der MMU (Memory Management Unit) vollzogene Speicherverwaltung nach dem "Demand-Paged"-Verfahren. Diese anforderungsgesteuerte, seitenorientierte Speicherverwaltung beschleunigt den Datentransfer zwischen ALU und Arbeits-/Massenspeicher durch eine höhere Effizienz bei der Nutzung von Arbeits- und Massenspeicher. Darüber hinaus unterstützt eine Fließkomma-Einheit (FPU = Floating Point Unit) die Verarbeitung umfangreicher arithmetischer Ausdrücke, so daß auch hierdurch eine weitere Entlastung der CPU stattfindet. Auf den Einsatz der Systeme übertragen bedeutet dies, daß DDP-Applikationen in Bereichen wie Versicherungen, öffentliche Verwaltung, Industrie, Handel, Banken sowie Bildung und Wissenschaft ohne Leistungseinschränkung beim Anschluß von bis zu acht Arbeitsplätzen möglich sind. Weitere Aspekte bei der Betrachtung der Verarbeitungsleistung sind der Einsatz von 256-KByte-RAMs aus der Siemens-Fertigung sowie von Festplatten-Laufwerken mit Kapazitäten (wahlweise) von 23,0, 36,8 oder 73,7 MByte (formatiert).

Den bei Mehrbenutzer-Betrieb problematischen Datenschutz hat der Systemhersteller durch Betriebssystem-Modifikationen gelöst. Der Zugang zum System wird durch Benutzer-Erkennung und Paßwort kontrolliert. Der unberechtigte Zugriff auf Datenbestände wird durch ein mehrstufiges Datensicherungssystem verhindert.

Sinix steht für Multiuser und Multitasking

Im Markt des Distributed Data Processing (DDP) steigen die Anforderungen der Anwender besonders hinsichtlich dezentraler Verarbeitungsleistung, hoher Verfügbarkeit, Kommunikations-Fähigkeit mit übergeordneten Systemen sowie einfacher Bedienung. Die Bedeutung des einzelnen Arbeitsplatzes innerhalb der DV-Struktur nimmt deshalb stetig zu. Der Anwender soll und möchte ohne tiefergehendes DV-Wissen an seinem Arbeitsplatz Planungsaufgaben lösen, Informationen in einer Datenbank speichern und von dort abrufen, Briefe schreiben und drucken sowie eine Vielzahl anderer Büro- und Verwaltungsarbeiten machen. Darüber hinaus soll es jedoch gleichermaßen möglich sein, neben der Standard-Anwender-Software auch selbstprogram-

mierte Lösungen auf einem dezentralen System einsetzen zu können.

Dieses Anforderungsprofil lag der Entwicklung zugrunde, die zur Sinix-MX-Familie auf der Basis des 32000-Chipsatzes führte. Als Betriebssystem, das diesen Anforderungen am besten entsprach, bot sich Sinix an, das kompatibel ist zu Xenix-V-3.0, welches wiederum auf der Basis des Unix-Systems-III entwickelt wurde. Als Timesharing-Betriebssystem hat Xenix-V-3.0 und damit auch Sinix am Weltmarkt eingeführte Schnittstellen, die eine Nutzung von weitverbreiteter Standard-Software ermöglichen. Unter dem Aspekt hoher Software-Kosten ist dieses Merkmal sicherlich wesentlich. Auch was den Hardware-Aufwand anbelangt, so bietet das Betriebssystem durch die gemeinsame Nutzung lokaler Verarbeitungs-Kapazität, gemeinsamer Datenbestände sowie von Peripherie-Geräten wie Druckern und Massenspeichern preisgünstige Lösungen. Für den Einsatz in Fachabteilungen großer Organisationen und Unternehmen ist darüber hinaus besonders ein gemeinsam genutztes Datenhaltungs-System von Bedeutung. Es gewährleistet, daß alle Nutzer mit den gleichen, konsistenten Daten arbeiten. Aufgrund der schnellen Ausbreitung von Unix-kompatiblen Systemen, wie z. B. Sinix, wächst sowohl die Software-Basis an, als auch das Potential ausgebildeter und eingearbeiteter Mitarbeiter, die mit Unix vertraut sind. Sinix ist ein portierbares Betriebssystem. Die auf Sinix geschriebenen Programme lassen sich mit begrenztem Aufwand an neue Hardware anpassen. Als Multitasking-System können unter Sinix mehrere Prozesse parallel laufen. Dadurch entfallen in vielen Applikationen Wartezeiten bei der Druckerausgabe und beim Filetransfer. Diese Aufgaben werden unter Sinix als Hintergrund-Prozesse ausgeführt. Der Anwender kann während dieser Zeit am Bildschirm weiterarbeiten.

Als Einstiegssystem in die Mehrplatz-Verarbeitung ist das PC-MX2 konzipiert. Das darauf implementierte PC-MX2.Sinix unterstützt bis zu sechs Arbeitsplätze. Es besteht aus:

- Menü-Shell (Benutzeroberfläche),
- Systemprogrammen und Kommandoprozeduren für Benutzer-Systemfunktionen,
- Betriebssystemkern mit Prozeßverwaltung, Dateisystem und Speicherverwaltung.

Der Kern beinhaltet alle Funktionen zur Ablaufsteuerung von System- und Anwendungsprogrammen. Die wesentliche Funktion des Sinix-Kerns ist die Steuerung der CPU 32016 und der Slave-Prozessoren für die virtuelle Speicherverwaltung, das Interrupt-Handling der Peripherie-Geräte sowie die Verwendung der Fließkomma-Einheit. Der Sinix-Kern ist im Hauptspeicher resident. Die anderen Systemkomponenten lassen sich bei Bedarf auf den Plattenspeicher auslagern. Alle weitere Software sind eigenständige Programme, die von der Prozeßverwaltung gesteuert und eingesetzt werden. Diese Prozeßverwaltung geschieht so, daß für jedes ablaufende Programm ein Prozeß erzeugt wird, der nun selber wieder Unterprozesse generieren und verwalten kann. Mit dieser Organisation ist die Basis für den Mehrbenutzer-Betrieb geschaffen. Für jeden Benutzer wiederum können mehrere Prozesse ablaufen (Multitasking). Jedem aktiven Terminal sind ein oder mehrere Dialog-Prozesse zugeordnet, mit deren Hilfe sich bei Bedarf mehrere Hintergrund-Prozesse starten lassen. Die vom System und den Anwendungsprogrammen benötigten Daten werden von einem Dateisystem mit baumartiger Struktur verwaltet. Dieses Dateisystem regelt unter anderem die Zugriffskontrolle über die Dateien und sichert die Geräteunabhängigkeit der Datenbasis. Die baumartige Struktur der Datenorganisation ermöglicht auch bei einer großen Zahl von Dateien den schnellen Zugriff auf einzelne Files. Das Betriebssystem bietet eine Reihe von Kommandos zur Dateiverwaltung, wobei die hierarchische Ordnung des Dateisystems genutzt werden kann. So lassen sich z. B. mit einem einzigen Kommando alle Dateien eines Verzeichnisses kopieren. Für jede Datei können differenziert Zugriffsberechtigungen vergeben werden. Diese Zugriffserlaubnis kann für den Eigentümer, die Benutzergruppe des Eigentümers und auch für alle anderen Benutzer gelten. Die Zugriffskontrolle läßt je nach Erfordernis das Lesen, das Schreiben oder die Ausführung von Programmen und Prozeduren zu. Der Sinix-Kern zwingt dem Inhalt einer Datei keine bestimmte Struktur auf. Dateien sind Sequenzen von Bytes. Die Peripherie-Geräte werden vom Betriebssystem als spezielle Dateien angesprochen. Die Definition der Peripherie-Geräte muß nicht bei der Programmierung erfolgen, sondern erst beim Ablauf des Programms. Die Plattendateien werden in Blöcken von 8 KByte bzw. in Fragmenten von 1 KByte verwaltet. Diese Blöcke sind im Hauptspeicher gepuffert (Cache-Methode). Höherstehende Zugriffsmechanismen, wie z. B. der indexsequentielle Zugriff, sind nicht im Sinix-Kern realisiert sondern Bestandteil der Anwendung.

Die Gerätetreiber des Betriebssystems sind maschinenspezifisch. Es werden zeichenorientiert arbeitende Bildschirme, verschiedene Drucker, 5,25-Zoll-Platten- und Floppy-Disk-Speicher, ¼-Zoll-Magnetband-Kassettenlaufwerke, eine batteriegepufferte Echtzeituhr mit nichtflüchtigem Speicher sowie ladbare Kommunikationsprozessoren unterstützt. Jeder Prozeß hat einen eigenen virtuellen Adreßraum von 16 MByte. Die maximale Programmgröße ist durch den für das Paging verfügbaren Plattenspeicher-Bereich von etwa 4 MByte begrenzt. Die Speicherverwaltungs-Einheit (Memory Management Unit) bildet die virtuellen Adressen auf den reellen Hauptspeicher mit maximal 4 MByte Kapazität ab. Durch diese Abbildung sind die einzelnen Prozesse voneinander geschützt, so daß eine hohe Betriebssicherheit des Systems gewährleistet ist. Überschreitet der Platzbedarf aller Prozesse den realen Hauptspeicher, so werden die aktuell nicht benötigten Speicherseiten auf die Platte geladen, um im Hauptspeicher Platz zu schaffen.

Für den Benutzer bietet das Betriebssystem zwei verschiedene Modi an:

- den Experten-Modus mit originalem Zugang zur Sinix-Shell durch explizite Angabe des Kommandos und Parameter.
- die Menü-Oberfläche zur Eingabe von Kommandos über die Menü-Shell. In diesem Modus wird nach der Eingabe entweder ein Folgemenü angezeigt, das Kommando direkt von der Menü-Shell ausgeführt, ein autonomes Programm gestartet oder eine Kommando-Prozedur abgearbeitet.

Weniger häufig benutzte Funktionen des Betriebssystems werden mit Hilfe von Kommandos in der Original-Unix-Kommandosprache zeilenorientiert eingegeben. Der Programmierer kann bei Bedarf eine Folge von Kommandos zu einer Prozedur und damit zu einem neuen Kommando zusammenfassen. Quellcodes von Programmen und Dateien lassen sich mit einem bildschirmorientierten Texteditor (CED) erstellen und warten. Dieser Editor ermöglicht eine freie Positionierung der Schreibmarke, freie Belegung der Funktionstasten mit Kommandos oder Texten, parallelen Direktzugriff auf mehrere Dateien, Bearbeitung ganzer Textzeilen, Texttabellen und Textblöcke, die Übergabe von Textsegmenten an Sinix-Kommandos sowie aufrufbare Hilfsinformationen und eine ständig angezeigte Befehlsübersicht.

Der Hauptspeicher-Bedarf für Sinix-V-2.0 (PC-MX2) beträgt je nach Speicherausbau 400 bis 800 KByte.

Herzstück des Hardware-Konzeptes: 32000

Bei der Konzeption einer für den Anwender zukunftssicheren Unix-Computer-Familie waren neben der Leistungsfähigkeit auch andere Kriterien von großer Bedeutung. Dazu zählen im besonderen Maße die Architektur der System-Prozessoren, die Ausbaufähigkeit der Prozessor-Familie, die Verfügbarkeit eines möglichst umfassenden Chipsatzes sowie die Vertrauenswürdigkeit und Erfahrung des Prozessor-Lieferanten. Im Falle der Sinix-PC-Familie von Siemens, bestehend aus den Systemen PC-MX2, für maximal sechs Arbeitsplätze, PC-MX4, für maximal 16 Arbeitsplätze und PC-2000 für maximal sechs Arbeitsplätze aber zusätzlicher Implementation des Betriebssystems BS2000, fiel deshalb die Entscheidung auf den 32000-Chipsatz von National Semiconductor. Die Entscheidung begründet Siemens in einer seiner Publikationen über das Hardware-Konzept des PX-MX2 folgendermaßen: „Eingesetzt wird einer der derzeit leistungsfähigsten, am Markt befindlichen und für Sinix am besten geeigneten 32-Bit-Mikroprozessoren, der NS32016 von National Semiconductor.“

Im Grundausbau umfaßt das System die Systemeinheit 200 mit Multibus, Prozessor mit MMU, Hauptspeicher von 1 MByte, Floppy-Disk-Massenspeicher mit 650 KByte formatierter Kapazität, Baugruppenträger für einen AFP-Anschluß-Zusatz (AFP = Alternierendes Flanken-Pulsverfahren, zur Übertragung von Daten über Telefon-Nebenstellennetze), die Fließkomma-Einheit (FPU) sowie den Basis-E/A-Prozessor. Zur Entlastung des Arbeitsprozessors dienen neben der MMU und der FPU auch die batteriegepufferte Echtzeituhr.

Als periphere Massenspeicher sind Festplatten-Laufwerke mit 23,0, 36,8 oder 73,7 MByte formatierter Kapazität und ein Magnetband-Streamer mit 45 MByte Speicherkapazität anschließbar.

Systemerweiterungen sind ein ladbarer DFÜ-Prozessor mit 128- oder 256-KByte-RAM, Speichererweiterungen um 1 oder 3 MByte, EA-Prozessoren für Datenstationen, Aufrüstungen für die Plattenlaufwerke sowie Nachrüstsätze für AFP. Bildschirm-Arbeitsplätze gibt es mit oder ohne AFP.

Im Hauptspeicher kommen moderne 256-KBit-Bausteine zum Einsatz. Der Grundausbau des Hauptspeichers beträgt 1 MByte und kann bis auf 4 MByte ausgebaut werden. Speicherprozesse im Hauptspeicher sind durch 1 Paritätsbit gesichert. 1-Bit-Fehler beim Speichern werden deshalb erkannt.

Für das System PC-MX4, mit Anschlußmöglichkeiten für 16 Arbeitsplätze, gelten als Verarbeitungsrechner die gleichen Merkmale, die vorab für das System PC-MX2 geschrieben wurden. Mit den zusätzlichen Funktionen jedoch kann der PC-MX4 besonders als Netzrechner eingesetzt werden, da der integrierte Kommunikations-Rechner Netzwerke steuern kann. Der Hauptspeicher dieses Gerätes läßt sich bis auf 8 MByte ausbauen. Die Plattenspeicher-Kapazität ist auf 160 MByte erweiterbar. Auf dem integrierten Kommunikations-Prozessor läuft das Kommunikations-Programm CCP simultan zum Betriebssystem Sinix im Verarbeitungs-Prozessor ab. Im Netzwerkbereich dient der PC-MX4 als Konzentrator und stellt dem Netz eine Untermenge von Transdata-Datenverarbeitungs-Funktionen zur Verfügung. Die Kommunikation auf großen Strecken erfolgt über Datex-L- und Datex-P-Netz sowie über normale Postleitungen. Unzureichende Verbindungen werden automatisch analysiert und neu aufgebaut.

Im PC-2000, einem Mehrplatzsystem mit Universalrechner-Charakter, arbeitet der Verarbeitungsprozessor 32016 mit einem BS2000-Befehlsprozessor zusammen, der für den Anwender die Möglichkeit schafft, Applikationsprogramme, die normalerweise nur auf den BS2000-Mainframes der Serie 7500 laufen, auch auf PC-Hardware zu nutzen. Mit diesem System für Sinix- und BS2000-Programme wird die Grenze vom PC zum Universalrechner quasi überschritten.

Zusammenfassung

Bei seiner Sinix-PC-Familie, die am oberen Rande des PC-Leistungsspektrums angesiedelt ist, hat sich die Siemens AG als zentrale Verarbeitungseinheit für den 32000-Chipsatz, bestehend aus CPU 32016, MMU 32082 sowie FPU 32081 entschieden. Der Grund für die Auswahl dieses Chipsatzes liegt in der nach sorgfältiger Evaluierung entstandenen Erkenntnis des Systemherstellers, daß es sich hierbei um einen der leistungsfähigsten 32-Bit-Prozessoren handelt.

Uwe Bannow

3D-Echtzeit-Grafikrechner mit 32-Bit-Unix-Prozessor

Trotz des Erscheinens leistungsfähiger Grafikprozessoren bieten weiterhin nur komplexe Systemarchitekturen Lösungen für die 3D-Echtzeitsimulation. Bei vertretbarem Aufwand eignet sich dafür besonders das Konzept der Vektormaschine. Zusätzlich werden die Grafik-Arbeitsplätze immer häufiger mit einem hohen Maß an lokaler Intelligenz für die Applikations-Software ausgestattet. Für die Integration in einen Arbeitsplatz wie das 3D-Grafiksystem „IMI555“ bietet sich das auf dem Chipsatz der Serie 32000 basierende Modul ICM-3216 an, das in den Leistungsbereich einer VAX vorstößt.

Die Entwicklung neuer Workstations darf heute nicht nur unter dem Aspekt der Grafik geschehen. Vielmehr hat sich mit den Unix-Implementationen System V und Berkeley 4.2 ein Standard auf fast allen intelligenten Grafikarbeitsplätzen etabliert. Schon in der Konzeptionsphase ist die Entscheidung zu treffen, ob ein neuer Unix-Mikrorechner zu entwickeln ist oder eine Lösung vom Markt eingekauft und integriert werden kann. Um die Kosten niedrig zu halten und sich ganz auf die Entwicklung der Grafikmaschine konzentrieren zu können, entschied sich Interactive Maschines Inc. (IMI) für die Integration eines modernen Mikrorechnerkonzeptes.

Besonders interessant erscheinen die Einplatinen-Rechner, die niedrigen Preis und hohe Leistung vereinigen. Im Gegensatz zu VME- oder Multibus-II-Rechnern benötigen sie keine Backplane und keine Buslogik – Speicherzugriffe sind ohne Wartezyklen möglich. Die Wahl fiel auf das ICM-3216 von National Semiconductor, dessen Konzept (*Bild 1*) neue und einzigartige Funktionen bietet.

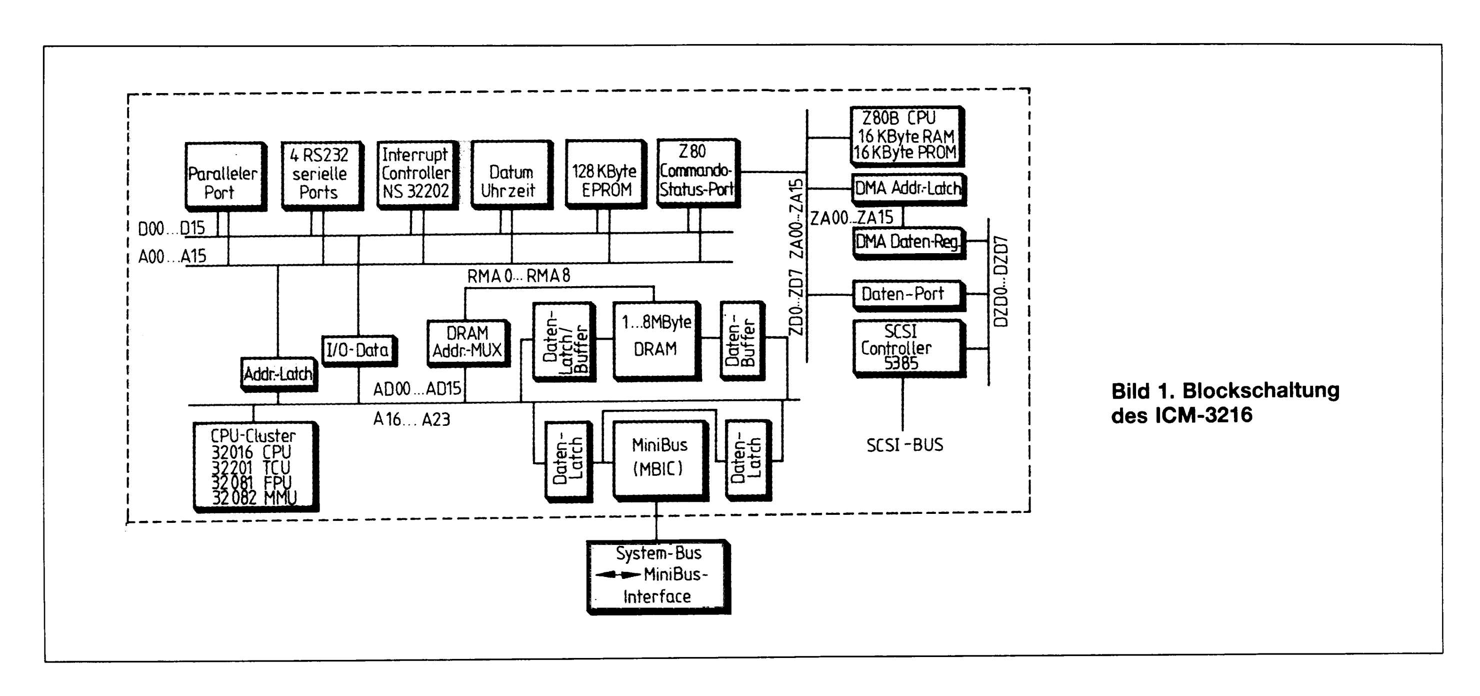


Bild 1. Blockschaltung des ICM-3216

Modular ohne Busplatine

Das ICM-3216 – Integrated Computer Module – vereinigt bis auf den Speicher alle Komponenten eines Unix-Rechners auf einem Modul, der CPU-Platine. Speicher-Module werden über einen Speicherbus auf die Rückseite der CPU-Platine aufgesteckt, wobei die Verbindung zwischen den Platinen als 96polige VG-Leiste ausgeführt ist. Mit zwei Speicher-Modulen können bis zu 8 MByte Arbeitsspeicher bei 10 MHz Taktfrequenz ohne Wartezyklus betrieben werden.

Kern des CPU-Moduls ist das CPU-Cluster, bestehend aus NS32016 CPU, NS32201 TCU (Timing Control Unit), NS32202 ICU (Interrupt Control Unit), NS32081 FPU (Floating Point Unit) und NS32082 MMU (Memory Management Unit). Bis zu fünf Peripheriegeräte wie Terminals, Drucker, Plotter, Joysticks, Trackballs oder Mäuse können über die vier RS-232C- und eine Centronics-Schnittstelle angeschlossen werden. Das Small Computer System Interface (SCSI) ist Schnittstelle für Festplatten, ¼"-Streamer und Floppy-Laufwerke. Eine Z80A CPU und ein NCR-5385-SCSI-Controller entlasten dabei den NS32016-Prozessor von der I/O-Steuerung.

Um weitere Peripherie an das ICM-3216 anschließen zu können, steht zusätzlich zum lokalen Speicherbus ein synchroner 16-Bit-Bus, der MiniBus, auf dem CPU-Modul zur Verfügung.

MiniBus

Das Interface zwischen CPU und MiniBus wird durch den MiniBus-Interface-Chip (MBIC), ein Gate Array NSC 6224, unterstützt. Der MBIC setzt Lese- und Schreibzyklen der CPU in MiniBus-Zyklen um, bedient Interrupts, generiert Wartezyklen zur Synchronisation der CPU mit dem MiniBus und stellt Steuersignale für die lokalen Speicher zur Verfügung (*Tabelle*).

Der MiniBus unterstützt synchrone 16-Bit- und Dual-Ported-Speicherzugriffe bei einer Taktfrequenz von 8 MHz und asynchrone 8-Bit- und 16-Bit-I/O-Operationen.

Die MiniBus-Signale

Mnemonic	Beschreibung
AD21–AD00	Address/Data/Control
BCOD0, BCOD1	Bus Code
BUSERR	Bus Error
BUSPAR	Bus Parity
BREQ0–BREQ7	Bus Requests
MBICCLK	MBIC Bus Clock
BCLK, BCLK2	System Bus Clocks
INT00–INT07	Interrupt Requests
IO	I/O Transfer
PFAIL	Power Fail
PWAIT	Peripheral Wait
RESET	Reset
RFRSHTIM	Refresh Interval

Im synchronen Betrieb hat er beim Schreiben eine Übertragungsrate von 5,3 MByte/s und beim Lesen von 4,0 MByte/s. Bis zu acht Bus-Master können am MiniBus betrieben werden. Der Adreßbereich des MiniBus umfaßt 4 MByte, die sich wie folgt aufteilen:
00 0000...3D FFFF: 16-Bit-Speicher-Bereich (synchroner Betrieb)
3E 0000...3E FFFF: 8-Bit-I/O-Bereich (256 Ports)

Diese 64 KByte des Adreßbereiches werden in asynchrone I/O-Zyklen übersetzt. Dabei enthalten AD15...AD08 die Portadressen und AD07...AD00 die Daten. Da Portadresse und Daten parallel mit einem Transfer übertragen werden, sind nur 256 Ports definiert.
3F 0000...3F 7FFF: 16-Bit-I/O-Bereich (16 384 Ports)

Diese 32 KByte des Adreßbereiches werden in asynchrone erweiterte I/O-Zyklen übersetzt. Mit AD14...AD00 wird der I/O-Port für den 16-Bit-Transfer adressiert.
3F 8000...3F BFFF: physikalischer Adreßbereich

Diese 16 KByte des Adreßbereiches sind für die Adressierung von Modulen, die eine vollständige Implementation des MiniBus besitzen, reserviert.
3F C000...3F FFFF: freier Adreßbereich

In vielen der gegenwärtigen Mikroprozessorsysteme sind die letzten 16 KByte des Adreßbereichs mit Interruptvektoren und Systemtabellen belegt. Da man bei der Definition davon ausging, daß der MiniBus auf die letzten 4 MByte eines Systems gelegt wird, sind die letzten 16 KByte des MiniBus nicht belegt.

Die 51 Signale des MiniBus sind wie der Speicher-Bus über 96polige VG-Leisten ausgeführt. Ein Subset des Busses (46 Signale) ist auf die Reihen A und C des VG-Steckers gelegt, damit der Anschluß von Modulen über ein 64poliges Flachbandkabel möglich ist. Dieses Design erspart eine Busplatine (die VG-Stecker und Leisten sind der eigentliche Bus) und kommt mit einem minimalen Aufwand an Steuerlogik aus – im Gegensatz zum VMEbus oder Multibus-II.

Bei der Integration in den Grafikrechner wurden nur die synchronen Lese- und Schreiboperationen und der asynchrone 8-Bit-I/O-Transfer des MiniBus implementiert. Das Interface zwischen Systembus und MiniBus (*Bild 2*) wurde mit einer T-State-Maschine realisiert, die zum MiniBus hin einen Subset des MBIC und zum Systembus einen Subset eines erweiterten IEEE-696-Busprotokolls zur Verfügung stellt. Über den MiniBus sind immer nur 2 MByte Seiten des Systembus direkt ansprechbar. Um den gesamten Speicherbereich des Systembus von 16 MByte adressieren zu können, wird auf dem Interface ein Register als I/O-Port vom MiniBus aus mit der Seitenadresse geladen. Aus den drei höchstwertigen Bits des Registers werden die Adreß-Signale AD21...AD23 für den Systembus generiert. Diese Implementation des MiniBus erlaubt es in den Display- und Refreshspeicher mit 3 MByte/s zu schreiben (*Bild 3*) und aus ihm mit 2,5 MByte/s zu lesen (*Bild 4*). *Bild 5* zeigt den asynchronen 8-Bit-I/O-Zyklus auf dem MiniBus.

Schnelle 3D-Grafik

Der Grafikrechner selbst enthält eine leistungsfähige Multiprozessor-Architektur mit Pipelining für die verschiedenen Aufgaben wie Darstellung, Refresh, Steuerung und Transformation.

Die Displayliste wird in einer PHIGS-ähnlichen Sprache und Struktur vom lokalen Mikroprozessor im 2 MByte großen Displayspeicher aufgebaut und verwaltet. In die Displayliste werden die dreidimensionalen Beschreibungen der Objekte mit den dazugehörigen Transformationen in Gleitkommaformat abgelegt. Für Transformationen und Bildmanipulationen stehen 3D-Befehle wie Rotation, Verschiebung, Perspektive, Mehrfachansicht, Maßstabsänderung, Clipping, Zoom, Depth Cueing, Sun Vector und Back-Face Hidden Line Rejection zur Verfügung.

Ein mikroprogrammierter bipolarer Bitslice-Prozessor, der Displayprozessor, arbeitet die Displayliste ab und legt die Bildvektoren im Refreshspeicher ab. Dabei steuert er zwei Spezialprozessoren, den Addierer/Subtrahierer und den Multiplizierer/Dividierer, die ebenfalls aus bipolaren Bitslices bestehen und mit einer Rechenleistung von 8 Mio. Gleitkomma-Operationen pro Sekunde die Geometriedaten der Objekte transformieren. Diese Kombination ermöglicht die Transformation von 200 000 3D-Vektoren pro Sekunde und eine dynamische Darstellung von komplexen dreidimensionalen Objekten.

Der 256-KByte-Refreshspeicher, in dem bis zu 32 000 transformierte Vektoren abgelegt werden, wird im Double-Buffered-Mode betrieben. In jeweils eine 128 KByte große Seite schreibt der Displayprozessor das neu berechnete Bild, während der Displaygenerator über den Refreshcontroller aus der anderen Seite Vektoren liest und den Random-Stroke-Monitor treibt. Sobald der Displayprozessor mit der Berechnung des neuen Bildes fertig ist, werden die Seiten umgeschaltet. Dadurch können beide Seiten unbeeinflußt auf den Speicher mit voller Geschwindigkeit zugreifen, und ein kontinuierlicher Bildrefresh ist gewährleistet. Der Refreshspeicher besitzt noch einen dritten Port zum Systembus. Über diesen Systembus-Port können die Vektoren, die als Endpunktpaare mit Farbinformation in ei-

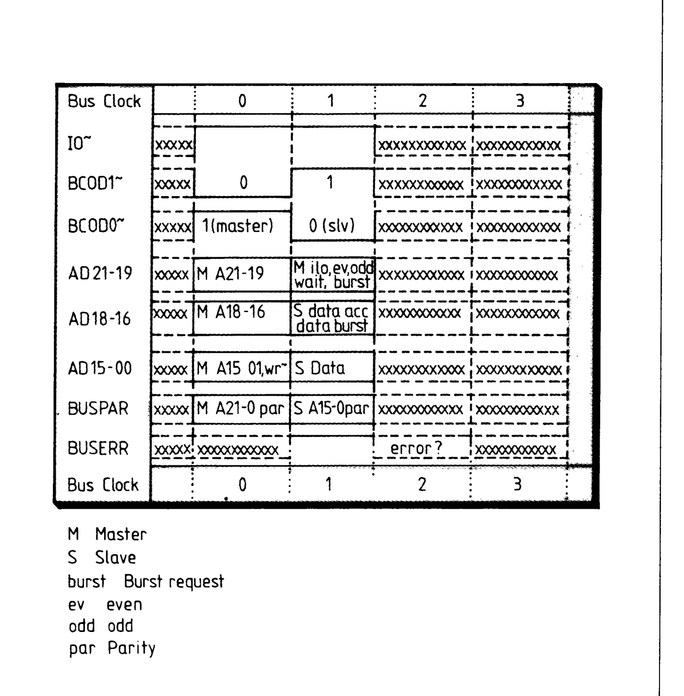


Bild 3. Synchroner Schreibzyklus auf dem MiniBus

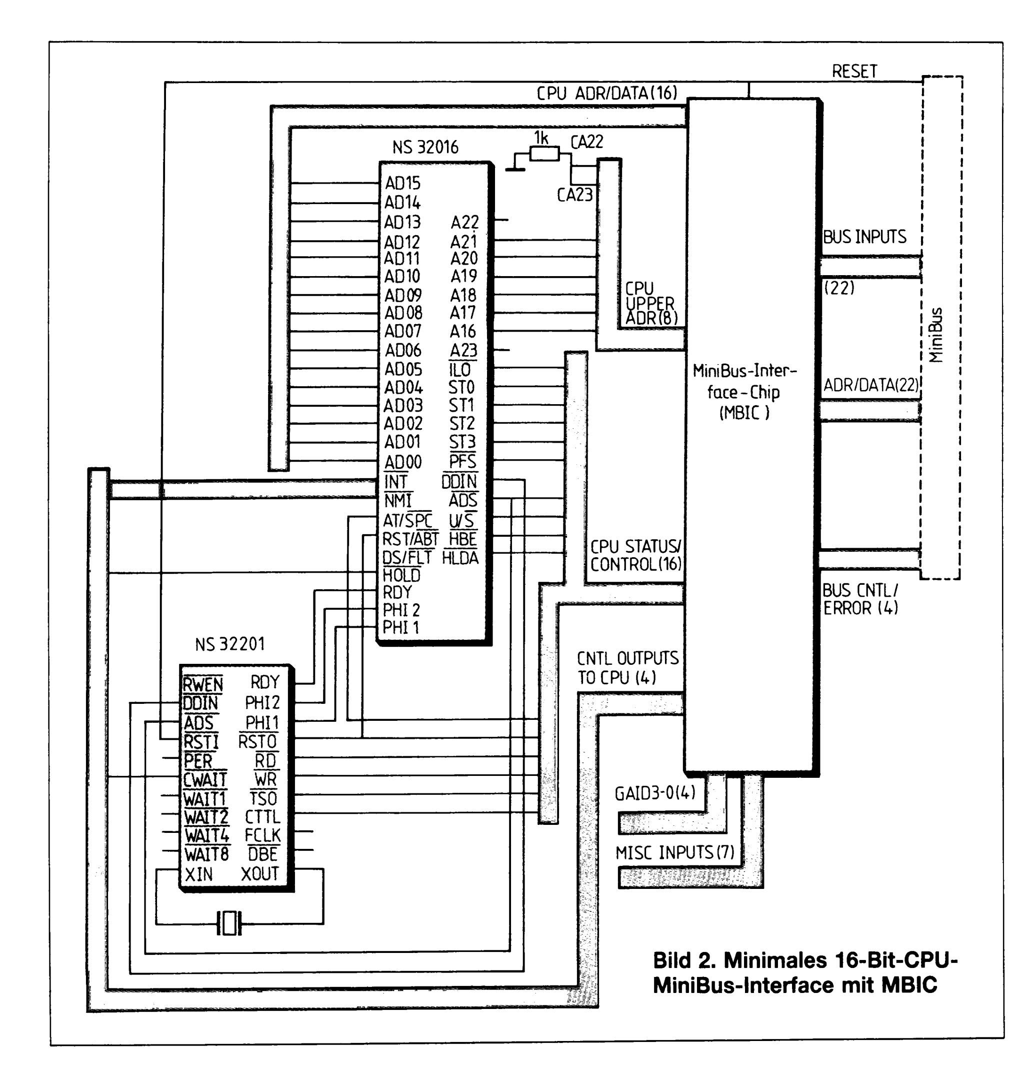
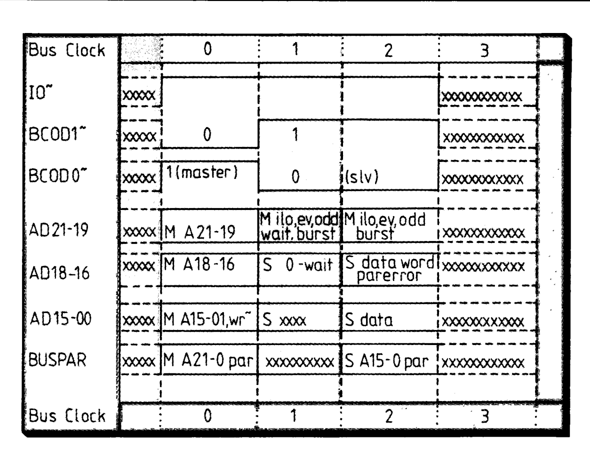


Bild 2. Minimales 16-Bit-CPU-MiniBus-Interface mit MBIC

nem 32-Bit-Wort abgelegt sind, vom lokalen Mikroprozessor aus gelesen und auf einen Plotter ausgegeben werden. *Bild 6* zeigt das Refreshspeicherformat bei Schwarzweißdarstellung, *Bild 7* bei Farbdarstellung.



M Master
S Slave
burst Burst request
ev even
odd odd
par Parity

Bild 4. Synchroner Lesezyklus auf dem MiniBus

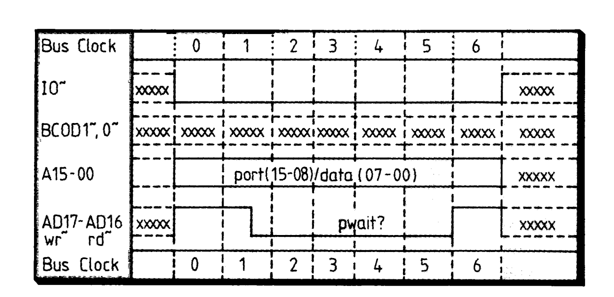


Bild 5. Asynchroner 8-Bit-I/O-Zyklus auf dem MiniBus

Cand. Ing. Uwe Bannow ist gebürtiger Hanseat und studiert Elektrotechnik an der TH Darmstadt. Sein Praktikum absolvierte er 1984 und 1985 bei der Fa. Interactive Machines Inc., CA, U.S.A. Dabei hat er u. a. bei der Integration des ICM-3216 in das 3D-Grafik-System IMI555 mitgearbeitet. Er ist Gesellschafter der imcos GmbH.



Da sich der Refreshspeicher als Buffer zwischen Displayprozessor und Displaygenerator befindet, ist der Bildrefresh nicht, wie bei vielen anderen Vektormaschinen, von der Rechengeschwindigkeit des Displayprozessors abhängig. Ein Bild muß immer erst dann neu berechnet werden, wenn die Displayliste geändert wurde. Selbst komplexe Darstellungen mit 32 000 Vektoren von 5 mm Länge werden flimmerfrei und mit einer konstanten Bildwiederholrate von 50 Hz gezeichnet. Der Displayprozessor trägt, nachdem er die Displayliste abgearbeitet hat, im letzten Refreshspeicherwort die Anzahl der darzustellenden Move- und Drawvektoren ein. Der Refreshcontroller übernimmt dieses Wort und übergibt die Wertepaare der Endpunkte an den Displayprozessor kontinuierlich. Die digitalen X-, Y- und Farb/Intensitätswerte werden vom Displaygenerator in analoge Signale umgewandelt. X- und Y-Wert sind 12 Bit lang. Damit können insgesamt 4096 × 4096 Punkte mit jeweils 128 Graustufen bzw. acht Farben mit jeweils 16 Intensitäten adressiert werden. Der Elektronenstrahl des Bildschirms wird direkt im Random-Stroke-Verfahren mit einer konstanten Geschwindigkeit von 14 mm/µs gesteuert.

Bis zu vier Farb- und Schwarzweiß-Monitore lassen sich gemischt gleichzeitig am Displaygenerator betreiben. Dabei können auf den Monitoren verschiedene Bilder oder parallel dasselbe Bild dargestellt werden. In *Bild 8* ist die Blockschaltung der Vektormaschine gezeigt.

31	30 24	23 12	11 0
move/ draw Flag	Intensität	y Vektor Wert	x Vektor Wert

Bild 6. Refresh-Speicherformat bei Schwarzweißdarstellung

31	30 27	26	25	24	23 12	11 0
move/ draw Flag	Intensität	r o t	g r ü n	b l a u	y Vektor Wert	x Vektor Wert

Bild 7. Refresh-Speicherformat bei Farbdarstellung

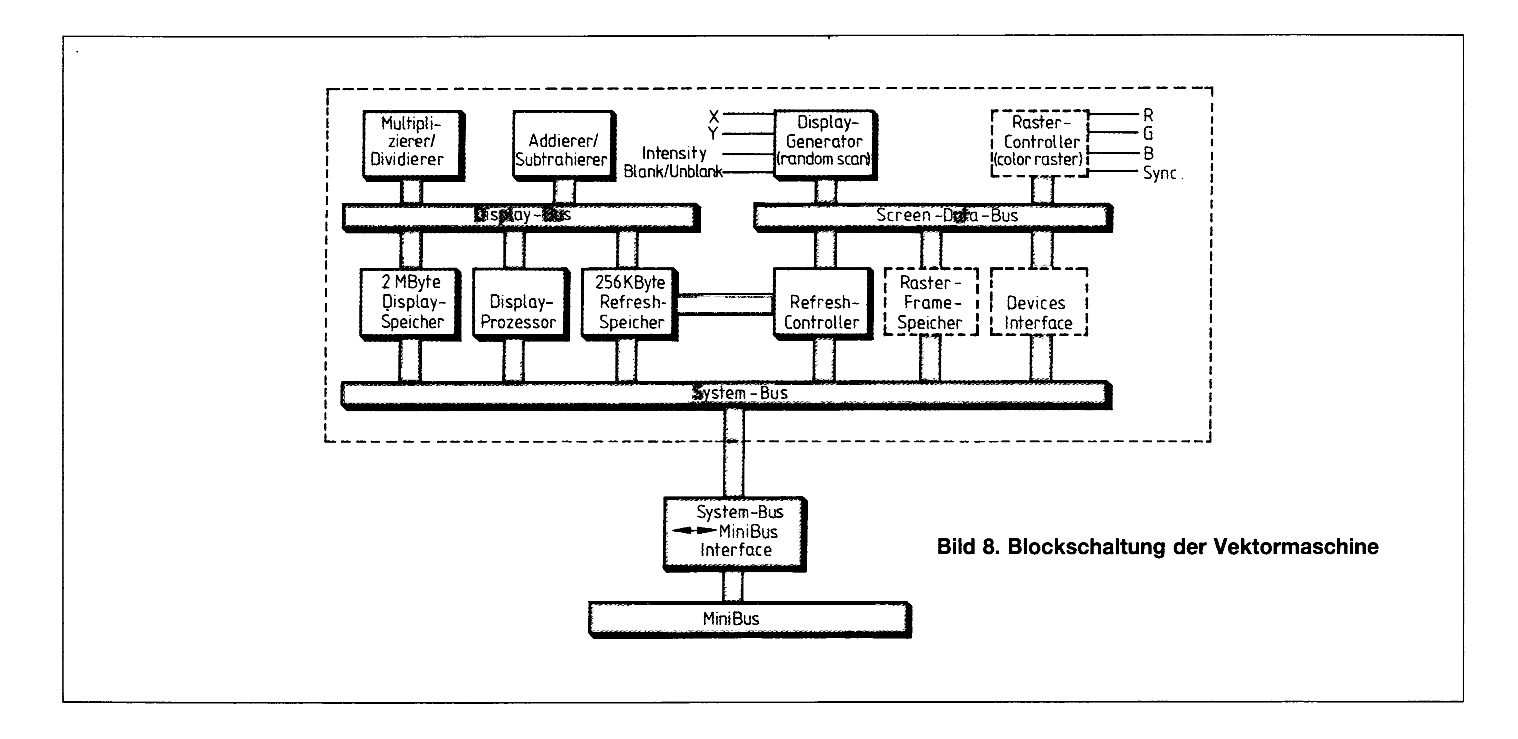


Bild 8. Blockschaltung der Vektormaschine

Anwendung

Vektormaschinen werden in der Luft- und Raumfahrtindustrie, bei Animationen in der Filmindustrie und in der Automobilindustrie für dynamische Echtzeit-Simulationen eingesetzt. Durch das ICM-3216 verfügt der Arbeitsplatz über eine große Massenspeicherkapazität – bis zu 500 MByte Magnetplatte und ein leistungsfähiges virtuelles Betriebssystem, Unix-System V. Die Applikationssoftware wird fast ausschließlich in „C" und Fortran 77 geschrieben. Der Floating-Point-Prozessor NS32081 stellt für die Anwendungsprogramme die Gleitkommarechenleistung zur Verfügung. Die zu animierenden bzw. zu simulierenden dreidimensionalen Objekte werden am Anfang mit den Grundtransformationen in die Displayliste eingetragen. Danach berechnet der lokale Mikrorechner nur noch die Parameter der einzelnen Transformationsmatrizen und trägt sie in der Displayliste ein. Die Grafikmaschine übernimmt dann die eigentliche Aufgabe der Objekttransformation. Um der Forderung nach Oberflächendarstellung zu entsprechen, arbeitet man an Raster-Frame-Speichern und Rastercontrollern, die direkt vom Displayprozessor gesteuert werden. Diese Kombination wird in Rastertechnik bei geringerer Auflösung ähnliche Geschwindigkeiten erreichen wie die Random-Scan-Technik.

Gary Fielland
Dave Roogers

Das „Pooled-Prozessor-Konzept"

Neue Wege in der Multi-Prozessor-Technik

Obwohl in der letzten Zeit durch moderne VLSI-Mikroprozessoren merkliche Leistungssteigerungen erreicht werden konnten, gibt es immer noch einen deutlichen Abstand zwischen Mikrocomputern mit einem Prozessor und den teureren Superminis, die aus bipolarer Logik und Bit-Slice-Prozessoren aufgebaut sind. Ein Weg, diese Lücke zu füllen, sind Systeme mit mehreren Prozessoren. Zu deren Realisierung gibt es unterschiedliche Konzepte, die je nach Anwendungsbereich die gewünschte Leistungssteigerung bieten. Ein Hauptproblem bei solchen Systemen besteht darin, daß mit wachsender Zahl von Prozessoren die Leistung nicht mehr gleichmäßig zunimmt, weil sich die CPUs gegenseitig behindern. Hier soll ein Konzept vorgestellt werden, das so angelegt ist, daß sich auch bei einer großen Zahl von Prozessoren in bezug auf die Verarbeitungsleistung kaum Schwierigkeiten ergeben.

Viele Prozessoren verderben die Leistung

Bild 1 zeigt das Spektrum der Systemkonzepte, die mit mehreren Prozessoren arbeiten. An einem Ende findet man die lose gekoppelten Konfigurationen, bei denen verschiedene Computer über serielle Verbindungen in einem Netzwerk verknüpft sind. Daneben gibt es unterschiedliche Arten eng gekoppelter Systeme. Eine Möglichkeit, die bei bestimmten Anwendungen sehr wirtschaftlich sein kann, ist die Aufteilung des Problems in genau definierte Tasks, die jeweils von einem spezialisierten Prozessor ausgeführt werden. Beispiele für diese „Applikations-Prozessoren" sind Funktionsblöcke für Video-Display-Steuerung, Sprachsynthese oder digitale Signalverarbeitung. Man verbindet diese Funktionseinheiten so, daß sie Daten und Kommandos vom Hostprozessor über den Systembus erhalten. Daneben besitzen sie eigene lokale Busse und lokale Speicher, um ihre Kommandos ausführen zu können. Bei einem solchen System handelt es sich um eine Master-/Slave-Konfiguration, wobei jeder Slave für eine spezielle Funktion zugeschnitten ist.

Obwohl ein solches Konzept in der Lage ist, sehr hohe Rechnerleistung zu realisieren, leidet es unter dem Nachteil der geringen Flexibilität. Jedes System muß an eine spezielle Aufgabe angepaßt sein. Eine wesentlich elegantere Lösung ist es, mehrere universelle Prozessoren mit gleichen Eigenschaften untereinander zu verknüpfen, wobei hier eine Einrichtung notwendig ist, die die Arbeitsbelastung unter den Prozessoren aufteilt.

Die einfachste Möglichkeit ist es in diesem Fall, jeden Prozessor mit seiner eigenen Kopie des Betriebssystems auszustatten sowie mit seinem eigenen Interrupt-System, lokalen Speicher und sogar jeweils einem Massenspeicher. Ein zentrales „Scheduling-Programm" teilt jeden neuen Job einem der Prozessoren zu. In der Praxis

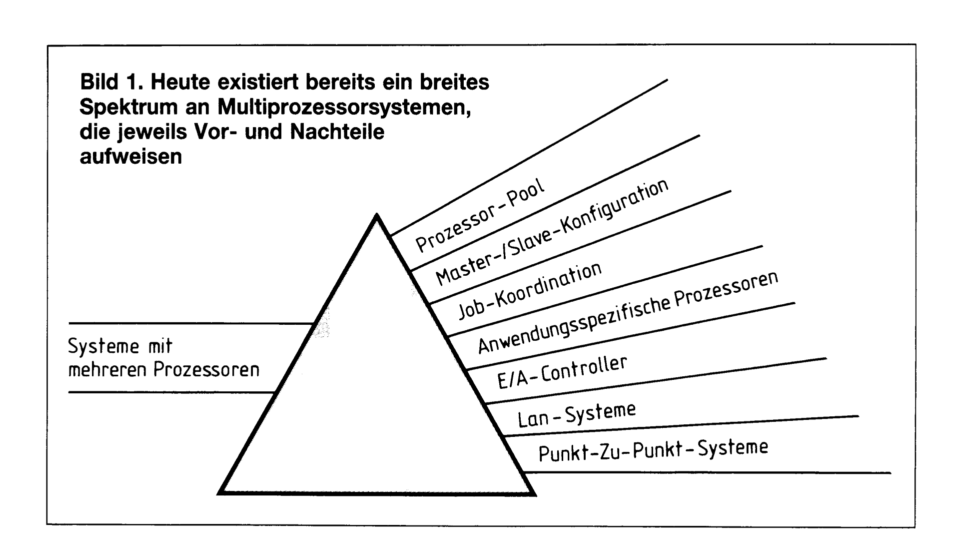


Bild 1. Heute existiert bereits ein breites Spektrum an Multiprozessorsystemen, die jeweils Vor- und Nachteile aufweisen

sieht das so aus, daß jeder Benutzer in einem Multi-User-System bei der Abarbeitung seines Jobs einem bestimmten Prozessor-Subsystem zugeordnet wird, das ihm für die Lösung seiner Aufgabe zugeteilt ist.

Obwohl es relativ einfach zu implementieren ist, hat auch dieses Konzept zwei prinzipielle Nachteile. Erstens wird jeder Job einem Prozessor zu Bedingungen zugeordnet, die zu Beginn der Abarbeitung vorliegen. Die Zuweisung läßt sich nicht mehr verändern, wenn die Bedingungen sich grundlegend ändern. Das kann beispielsweise zu Situationen führen, bei denen ein Prozessor überlastet ist, während andere Prozessoren nichts zu tun haben, weil das betreffende Problem schon gelöst ist. Zweitens ist es nicht möglich, die Abarbeitung eines rechenintensiven Problems durch Parallelverarbeitung zu beschleunigen.

Eine größere Effizienz läßt sich durch engeres Koppeln der Prozessoren in einer Master-/Slave-Konfiguration erreichen. Der gesamte Speicherbereich ist für alle Prozessoren zugänglich, allerdings ist ein Prozessor – in der Regel mit Hilfe von Software – als Master definiert. Dieser ist zuständig für Funktionen auf Systemebene, zum Beispiel für die Speicherverwaltung und die Steuerung der Zuweisung von Tasks für die Slave-Prozessoren. Der Master ist der einzige Prozessor, der in der Supervisor-Betriebsart laufen darf, die Slave-Prozessoren arbeiten nur in Benutzer-Betriebsart.

Wenn ein Slave-Prozessor auf Supervisor-Ebene arbeiten muß, beispielsweise um Exception-Traps zu behandeln, muß er diese Aufgabe dem Master-Prozessor übertragen. Eine solche Spezialisierung des Master-Prozessors führt zu einem potentiellen Engpaß, der die Systemeffizienz begrenzt. Daher eignet sich dieses Konzept lediglich für eine kleine Zahl von Prozessoren.

Bild 2. Grundaufbau der Prozessor-Pool-Architektur

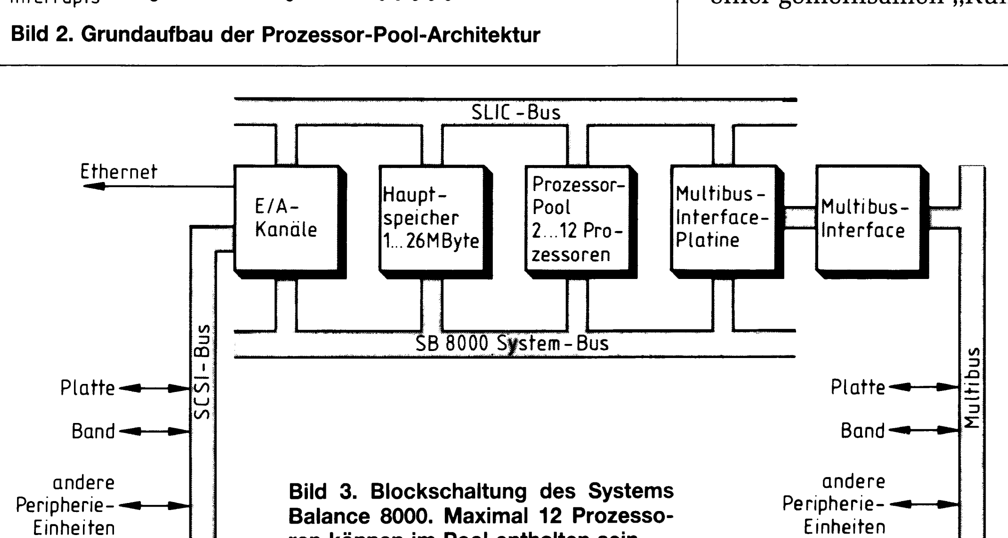


Bild 3. Blockschaltung des Systems Balance 8000. Maximal 12 Prozessoren können im Pool enthalten sein

Optimal: der Prozessor-Pool

Offensichtlich ist ein vollständig symmetrisches System die optimale Lösung, bei der alle Prozessoren gleichrangigen Zugriff auf den Systemspeicher, die Interrupts, E/A- und andere Ressourcen haben, so daß jede Task von jedem Prozessor auszuführen ist. Man kann dieses System als einen Pool von Prozessoren ansehen, die ein gemeinsames Betriebssystem sowie einen gemeinsamen Speicher usw. miteinander teilen. Ein zentraler Prozeß-Scheduler ist nicht erforderlich. Jeder Prozessor wählt für seine Arbeit die Prozesse aus einer gemeinsamen „Run-Queue“, die sich im allgemein zugänglichen RAM befindet. Eine solche Prozessor-Pool-Architektur ist außerordentlich flexibel, wenn die Anzahl der Prozessoren ein Parameter ist, den der Benutzer festlegen kann. Hier spricht man von einer SPP-Architektur („Scalable Processor Pool“, *Bild 2*).

Die SPP-Architektur hat wichtige Vorteile gegenüber anderen Multiprozessor-Konzepten. Weil sie symmetrisch aufgebaut ist, kann kein einzelner Prozessor zum Engpaß werden. Nach außen erscheint das System wie ein Einzelprozessor, die Systemleistung läßt sich, wenn es notwendig wird, durch Hinzufügen weiterer Prozessoren zum Pool erhöhen. Teure Systemressourcen werden von allen Prozessoren gemeinsam benutzt; Parallelverarbeitung läßt sich problemlos implementieren. Bis heute gab es allerdings einen wesentlichen Grund, der die Realisierung eines solchen Konzeptes verhinderte: Große Komplexität und hoher Preis ließen die Realisierung dieses Konzeptes unwirtschaftlich erscheinen.

Geändert hat sich diese Situation inzwischen, weil moderne 32-Bit-Mikroprozessoren als VLSI-Bauelemente zur Verfügung stehen, die nicht nur wesentlich leistungsfähiger als frühere Chips sind, sondern auch Funktionsmerkmale für die Systemebene bieten, zum Beispiel virtuelle Speicherverwaltung mit Demand-Paged-Zugriff und Unterstützung der Fließkomma-Mathematik. Die Familie 32000 bietet beispielsweise alle diese Merkmale in drei VLSI-Chips, die es ermöglichen, daß zwei komplette Prozessor-Subsysteme auf einer einzigen Platine unterzubringen sind. Diese Funktionseinheiten sind die Basis des Systems „Balance 8000“ von Sequent, bei dem es sich um das erste kommerziell erhältliche SPP-System handelt.

Bild 3 zeigt die Blockschaltung des Balance-Systems. Der Prozessor-Pool besteht aus 2...12 Prozessoren, die mit einem synchronen 10-MHz-Systembus arbeiten, bis zu 28 MByte Primärspeicher und leistungsfähigen E/A-Interfaces. Eine verbesserte Version des Betriebssystems UNIX, Berkeley 4.2 bsd, mit der Bezeichnung „DYNIX“ übernimmt die Systemverwaltung. Wesentliche Verbesserung ist die transparente Prozessor-Pool-Verwaltung. Das System besitzt auch einen Diagnoseprozessor.

Eines der wichtigsten Merkmale des Balance 8000 ist die automatische gleichmäßige Verteilung der Arbeitsbelastung, das heißt die dynamische Zuweisung der Tasks zu Prozessoren im Pool, was für den Benutzer und Programmierer völlig unsichtbar erfolgt. Daher verhält es sich wie ein Ein-Prozessor-System, wobei sich allerdings die Verarbeitungsleistung fast linear durch Hinzufügen zusätzlicher Prozessoren erhöhen läßt. Unabhängig von der Anzahl der Prozessoren muß lediglich eine Kopie des Betriebssystems im Speicher abgelegt sein.

Natürlich erfordert die Implementierung einer Hochleistungs-SPP-Architektur mehr als das Warten auf ausreichend leistungsfähige Mikroprozessorfamilien. Es gibt fünf wichtige Bereiche, die sorgfältig zu optimieren sind, um alle Vorteile der SPP-Architektur nutzen zu können. Es handelt sich um das Prozessor-Subsystem, den Systembus, das System-RAM, die E/A-Mechanismen sowie das Betriebssystem.

Prozessor-Subsystem

Die Grundlage eines jeden Prozessors sind drei VLSI-Bauelemente. Eine 32-Bit-CPU (NS32032), ein Fließkomma-Coprozessor (NS32081) sowie eine Speicherverwaltungseinheit (NS32082). Weil die meisten technischen Anwendungen sowie das Betriebssystem selbst häufig mit 32-Bit-Zahlen und -Adreß-Pointern arbeiten, sorgt eine interne und externe Wortbreite von 32 Bit für eine äußerst effiziente Programmausführung. Die Fließkommaeinheit arbeitet als transparente Erweiterung der CPU, so daß schnelle Berechnungen, die für die meisten technischen Anwendungen heute erforderlich sind, sehr preiswert unterstützt werden.

Die Speicherverwaltungseinheit arbeitet ebenfalls als transparente Erweiterung der CPU, wobei eine virtuelle Speicherverwaltung mit Demand-Paged-Zugriff implementiert ist. Hiermit ist es möglich, daß jeder Prozeß, der auf dem Balance 8000 abläuft, über einen logischen Adreßbereich von bis zu 16 MByte verfügt, das ist der vollständige Adreßbereich von 24 Bit der CPU 32032. Dieser ist in 32 768 Seiten zu jeweils 512 Byte aufgeteilt. Wenn ein Prozeß auf eine Seite zugreifen möchte, die sich derzeit nicht im RAM befindet, benutzt die MMU eine 2-Ebenen-Seiten-Umsetzungstabelle, um die erforderliche Seite aus dem Sekundärspeicher zu holen und sie in dem gewünschten RAM-Bereich zu plazieren.

Es ist außerordentlich wichtig, in einem eng gekoppelten Multiprozessorsystem, den Busverkehr und die Hauptspeicherzugriffe zu minimieren. Wenn verschiedene Prozessoren dauernd um den Zugriff auf den gemeinsamen Speicher „kämpfen“ müssen, kann die Systemleistung merklich zurückgehen. Es gibt eine „Daumen-Regel“, die besagt, daß jeder zusätzliche Prozessor lediglich 80 % der Leistung des Prozessors bringt, der davor hinzugefügt wurde. Dies bedeutet, daß das Hinzufügen eines zweiten Prozessors die Leistung um

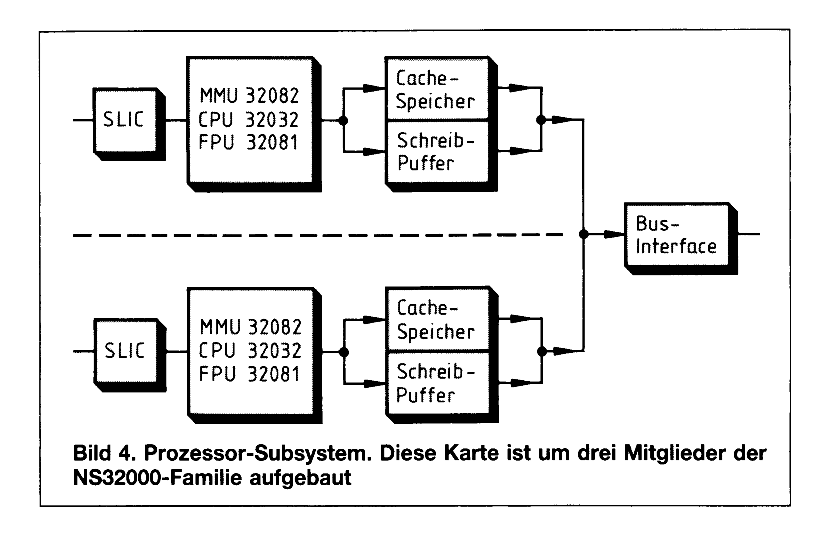


Bild 4. Prozessor-Subsystem. Diese Karte ist um drei Mitglieder der NS32000-Familie aufgebaut

80 % erhöht, so daß sich insgesamt eine Leistung von 180 % ergibt. Das Hinzufügen eines dritten Prozessors bringt zusätzliche 64 % (80 % von 80 %), so daß man jetzt eine Leistung von 244 %, also nur 36 % mehr als zwei Prozessoren, erhält.

Wenn man jetzt einen vierten Prozessor hinzufügt, trägt dieser 51 % (80 % von 64 %) zur Systemleistung bei, was eine effektive Steigerung um nur 21 % bedeutet. Ein fünfter Prozessor erhöht die Leistung um lediglich 14 %, so daß weitere Prozessoren offenbar keine wesentliche Leistungssteigerung mehr zulassen. Nach dieser „Daumen-Regel“ hätte ein System mit 12 Prozessoren eine effektive Leistung von 4,6 Prozessoren. In der Praxis würde sogar dieser geringe Wert nicht erreicht, weil der Systembus sehr schnell überlastet wäre, was wiederum dazu führt, daß die effektive Gesamtleistung sogar abnimmt.

Um einen möglichst großen Nutzen aus einem Pool von bis zu zwölf Prozessoren ziehen zu können, muß sowohl mit der Bus- als auch mit der Speicher-Bandbreite vorsichtig umgegangen werden, was man damit erreichen kann, daß man jeden Prozessor mit einem lokalen Cache-Speicher ausstattet. Dieser umfaßt 8 KByte und hat sehr kurze Zugriffszeiten. Hier sind die Inhalte der zuletzt abgefragten Hauptspeicherplätze abgelegt. Viele Untersuchungen haben gezeigt, daß während des größten Teils der Zeit Prozessoren mit der Behandlung von Daten oder Ausführung von Befehlen beschäftigt sind, die sich in einem relativ kleinen Adreßbereich befinden. Wenn der Prozessor auf einen Platz im Hauptspeicher zugreifen muß, untersucht man die Adresse zunächst daraufhin, ob sich die erforderlichen Daten nicht schon im Cache-Speicher befinden. Wenn dies der Fall ist, liefert der Cache die Daten, und ein Zugriff auf den Hauptspeicher ist nicht erforderlich.

Bild 5. Aufbau des Cache-Speichers. Dieser Hilfsspeicher sorgt dafür, daß die Busbelastung in erträglichen Grenzen bleibt

Der Cache im Balance 8000 ist in Zeilen zu je 8 Byte organisiert. Dies ergibt den besten Kompromiß zwischen Trefferverhältnis und Busbelastung; außerdem sind 8-Byte-Transfers besonders wichtig für den Systembetrieb. Simulationen und Messungen haben gezeigt, daß bei diesen Cache-Speichern ein Trefferverhältnis von 95 % zu erreichen war. Der Einfluß auf die Gesamtleistung ist merklich: Standard-Benchmarks zeigen, daß der Pool aus 12 Prozessoren die effektive Leistung von 11 Prozessoren erreicht, bei den meisten praktischen Anwendungen mit Rechentätigkeit sollte eine effektive Leistung von 8...11 Prozessoren zu erreichen sein.

Die Konzipierung eines Cache-Speichers für eine Prozessor-Pool-Architektur unterscheidet sich sehr stark von Cache-Konzepten für einen einzelnen Prozessor. Erstens sind alle Cache-Inhalte Kopien von Speicherinhalten im RAM, was bedeutet, daß bei Veränderungen der Cache-Daten auch das Original zu modifizieren ist. Im statistischen Mittel verbringen Prozessoren etwa 10...15 % ihrer Zeit damit, Schreibzyklen auszuführen. Dies bedeutet, daß in einem Multiprozessorsystem die Wahrscheinlichkeit groß ist, daß zwei oder mehr Prozessoren gleichzeitig darauf warten, den Hauptspeicher zu aktualisieren.

Um eine Leistungseinschränkung zu vermeiden, die sich ergibt, wenn die Prozessoren darauf warten müssen, den Speicher zu aktualisieren, besitzt der Balance 8000 Schreibpuffer. Wenn ein Prozessor in den Hauptspeicher schreiben muß, gibt er Adressen und Daten in den Schreibpuffer und fährt mit der Ausführung seines Programmes fort. Die Schreibpuffer-Schaltung wartet, bis der laufende Hauptspeicherzyklus beendet ist, greift dann auf den Bus zu, um die Daten unter der spezifizierten Adresse abzulegen.

Jeder Cache-Speicher besitzt auch eine zusätzliche Schaltung, die den Systembus überwacht, d. h. die die Schreibzyklen anderer Prozessoren beobachtet. Wenn ein anderer Prozessor den Inhalt eines Speicherplatzes verändert hat, der mit dem eigenen Cache-Inhalt übereinstimmt, sorgt diese Schaltung dafür, daß der Cache-Speicher an dieser Stelle ebenfalls aktualisiert wird. Dieser Mechanismus stellt sicher, daß nur gültige Daten im Cache-Speicher enthalten sind.

Jedes Prozessor-Subsystem enthält eine kundenspezifische Schaltung mit der Bezeichnung „SLIC“ (System Link and Interrupt Controller). Auch jeder Speichercontroller, E/A-Kanal und Bus-Koppler verfügt über jeweils einen SLIC. Alle SLIC-Bausteine kommunizieren über einen speziellen SLIC-Bus, der ein einfaches Kommando-Antwort-Paket-Protokoll benutzt. Ein SLIC dient drei Zwecken: Er stellt ein allgemeines Interrupt-System für den Prozessor-Pool dar, einen Semaphore-Cache sowie einen zusätzlichen Kommunikationspfad zwischen den Modulen.

Der konventionelle priorisierte Interrupt-Mechanismus von Einchip-Prozessoren ist für Multiprozessor-Architekturen nicht brauchbar. In einem echten SPP-System weiß man nicht, welcher Prozessor gerade einen bestimmten Prozeß abarbeitet oder welcher Prozessor einen bestimmten Interrupt bedient. Im Balance-System werden daher Interrupt-Requests in SLIC-Pakete umgeformt und auf dem SLIC-Bus im System ausgesendet. Die SLICs, die mit den Prozessoren verbunden sind, arbitrieren untereinander, indem sie die Prioritäten der Prozesse auf ihren eigenen Prozessoren prüfen. Der SLIC mit dem Prozeß entsprechender Priorität setzt das Kommandopaket in ein Interrupt-Request für den Prozessor um. Auf diese Weise wird die Bedienung von Interrupt-Requests dynamisch dem entsprechenden Prozessor

zugeordnet, der den Prozeß mit der geringsten Priorität abarbeitet.

Neben einem optimierten Mechanismus zur Behandlung von Hardware-Interrupts können von den SLICs auch Software-Interrupts auf unterer Ebene ausgesendet werden. Diese Einrichtung benutzt das Betriebssystem.

Der Cache für 1-Bit-Semaphoren in jedem SLIC stellt einen einfachen Ausschluß-Mechanismus dar. Das Betriebssystem benutzt diese Funktion zur Implementierung eines Ausschluß- und Synchronisations-Mechanismus auf höherer Ebene. Einer der Gründe für die Leistungsbegrenzungen in einigen früheren Mikroprozessorsystemen ist der selbstblockierende Zugriff auf einen Semaphore. Dies kann beim SLIC nicht passieren, denn SLICs sind an den lokalen Bussen und nicht am Haupt-Systembus angeschlossen.

Der SLIC-Bus wird auch bei Diagnose- und Debugging-Routinen benutzt. Während der Einschaltroutine testen die SLICs die Anwesenheit von Modulen, um sie optimal konfigurieren zu können. Ein Benutzer muß dem System nichts über die Hardware mitteilen, außerdem haben die Karten keine Schalter oder Drahtbrücken zur Identifizierung. Jede Karte, die ins Chassis eingesteckt ist, wird automatisch erkannt und konfiguriert.

Die SLIC-Schaltung auf der Speicher-Controller-Karte wird zur Weitergabe von Fehlerinformationen an das Betriebssystem benutzt. Auch diese Funktionseinheit gestattet es der Auto-Konfiguration-Software, Attribute zu setzen, z. B. die Basisadresse.

Bild 6 zeigt die Blockschaltung des SLIC, das als CMOS-Gate-Array in 3-µm-Technologie hergestellt wird. Beim SLIC-Bus handelt es sich um einen bitseriellen 2-Draht-Bus mit verdrahteter ODER-Verknüpfung und selbst-arbitrierendem Zugriffsmechanismus, ähnlich wie beim seriellen Teil des VMEbus. Ein SLIC beginnt mit der Übertragung des Paketes, wobei jedes Mal, wenn eine Kollision mit einem anderen SLIC festgestellt wird, die Einheit mit der geringsten Priorität ihre Übertragung abbricht, um sie später erneut zu versuchen.

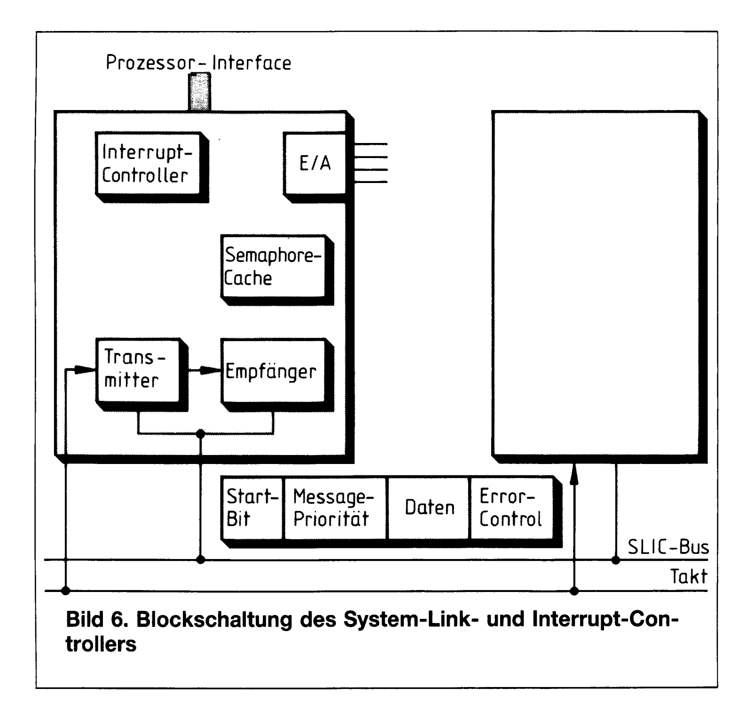


Bild 6. Blockschaltung des System-Link- und Interrupt-Controllers

Systembus

Es wurden mehrere moderne Techniken herangezogen, um einen Systembus für den Balance 8000 zu konzipieren, der für die hohe erforderliche Leistung einer SPP-Architektur ausgelegt ist, ohne zu große Kosten zu verursachen. Alle Prozessor-Subsysteme sind symmetrisch mit allen Ressourcen über einen synchronen 10-MHz-Bus verbunden, der über einen parallelen 32-Bit-Multiplex-Adressen/Daten-Pfad sowie eine Gruppe von Steuerleitungen verfügt.

Trotz des Zeitmultiplexverfahrens ist der Bus in der Lage, die geforderte Leistung zu erreichen, weil Hochgeschwindigkeits-TTL-Bauelemente vom FAST-Typ und ein Mehrfach-Pipeline-Paket-Protokoll für den Datenpfad zur Anwendung kommen. Separate Pipes werden für Schreib- und Lese-Operationen im Primärspeicher sowie für E/A-Aufrufe benutzt.

Wichtigstes Merkmal des Busprotokolls ist die Trennung zwischen Anfragen und Antworten. In einem konventionellen Mikroprozessor-Lesezyklus gibt die CPU eine Speicheradresse aus, wartet darauf, daß der Speicher die Daten ausgibt, und liest sie anschließend. Dieses einfache Protokoll ist für ein Hochleistungs-Multiprozessor-System nicht brauchbar, insbesondere dann, wenn Prozessoren auf langsamere Subsysteme über Multibus-Koppler zugreifen. Beim Busprotokoll des Balance 8000 hängt die Busbelegung nicht von der Zugriffszeit des Speichermediums ab, wie das im in *Bild 7* gezeigten Beispiel zu erkennen ist.

Diese Darstellung zeigt eine typische Momentaufnahme der Busaktivitäten, bei der drei Prozessoren den Bus für den Zugriff auf den Hauptspeicher benutzen und ein langsameres Subsystem über einen Multibus-Koppler angeschlossen ist. Es ist zu beachten, daß sowohl der Hauptspeicher als auch der Multibus-Koppler Frage- und Antwort-Warteschleifen besitzen. Zunächst gibt Prozessor A eine Anfrage zum Lesen eines Bytes aus der Multibus-Funktionseinheit aus und diese Anfrage gelangt in die Multibus-Request-Queue. Ohne auf die Antwort zu warten, gibt Prozessor A den Bus wieder frei, und die Prozessoren B und C geben direkt danach 8-Byte-Lese-Requests an den Hauptspeicher aus. Diese werden in die Warteschlange des Hauptspeichers eingefügt, so daß der Bus für weitere Aktivitäten frei ist. Wenn der Speichercontroller die Daten, die von Prozessor B gefragt wurden, vorliegen hat, dauert es zwei Buszyklen, um die 8 Bytes zu übertragen, danach zwei weitere, um die 8 Bytes, die vom Prozessor C gefragt wurden, zu transferieren. Der Bus ist dann wiederum frei, bis der Multibus-Koppler die von Prozessor A angefragten Daten für die Übertragung vorliegen hat. Aufgrund dieser Trennung von Anfragen und Anworten erreicht man beim Bus des Balance 8000 eine Bandbreite

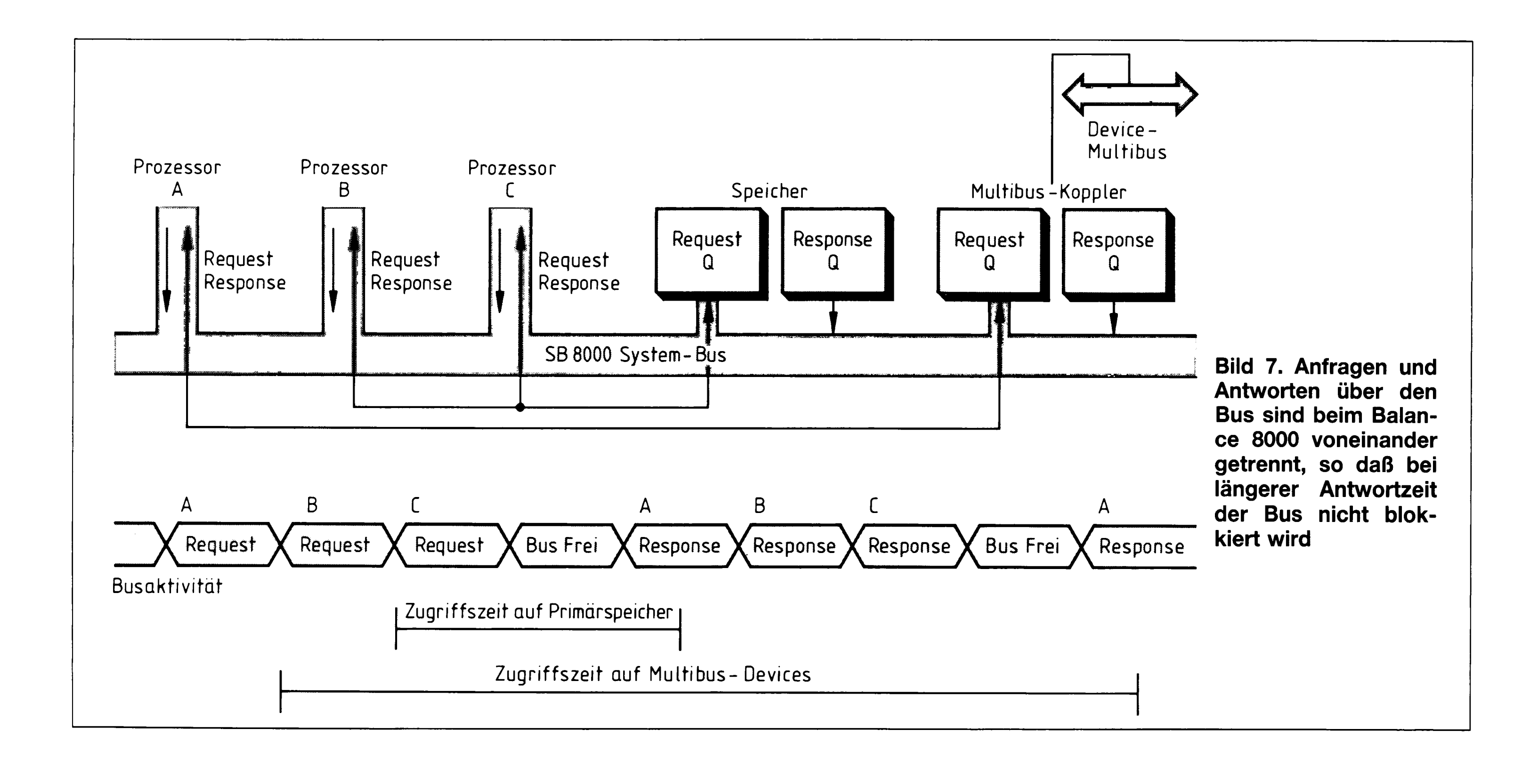


Bild 7. Anfragen und Antworten über den Bus sind beim Balance 8000 voneinander getrennt, so daß bei längerer Antwortzeit der Bus nicht blokkiert wird

von über 26 MBytes/s, wobei der theoretische maximale Wert bei 40 MBytes/s liegt.

Bandbreite ist nicht der einzige Gesichtspunkt für die Auslegung eines Multiprozessorsbusses. Arbitration, Kollisions-Management und Fehlerbehandlung erfordern mehr Aufmerksamkeit, als dies bei konventionellen Ein-Prozessor-Systemen der Fall ist. Beim System Balance 8000 verwaltet ein zentraler Arbiter mehrere Prioritätsebenen und garantiert, daß kein Prozessor vom Buszugriff abgeschnitten wir, auch wenn dieser sehr stark belastet ist. Damit wird nicht nur verhindert, daß das System „aussteigt“, sondern auch die Leistung von Peripherieeinheiten, wie zum Beispiel Festplatten, deren Datentransfers möglichst direkt erfolgen sollen, wird gesteigert. Ein potentielles Problem mit Busprotokollen, bei denen Anfrage und Antwort getrennt sind, ist die Möglichkeit, daß ein plötzliches starkes Anwachsen des Busverkehrs die Anfrage-Warteschlangen überfüllt. Ein Überlauf ist allerdings nicht zulässig, deshalb muß man einen Mechanismus zur Flußkontrolle vorsehen. Bei manchen Systemen erfolgt das in Form eines negativen Quittungssignals (NAK-Negative Acknowledgement), dieses Konzept führt aber zur Leistungsverminderung bei hoher Belastung, weil hiier nichtproduktive Zyklen einzufügen sind. Beim Balance 8000 erfolgt die Flußkontrolle dezentral, wobei jeder Prozessor oder Bus-Koppler Statusinformationen über jede wichtige Warteschlange erhält, um so sicherzustellen, daß Anfragen nur dann auf den Bus ausgegeben werden, wenn es einen freien Platz in der Ziel-Warteschlange gibt. So wird wird verhindert, daß Buszyklen für abgewiesene Zugriffanfragen verschwendet werden.

Für die Fehlerbehandlung enthält der Bus Paritäts- und andere Fehlerkennungseinrichtungen. Wenn ein ernsthafter Fehler erkannt wurde, zeichnet das System die Quelle des Fehlers auf, der Zustand des Systembusses wird eingefroren, und der Diagnoseprozessor übernimmt die Steuerung. Dieser nutzt die Vorteile des zusätlichen Kommunikationspfades, der sich mit dem SLIC-Bus ergibt, um das System zu prüfen und eine Reparaturaktion auszufüh-

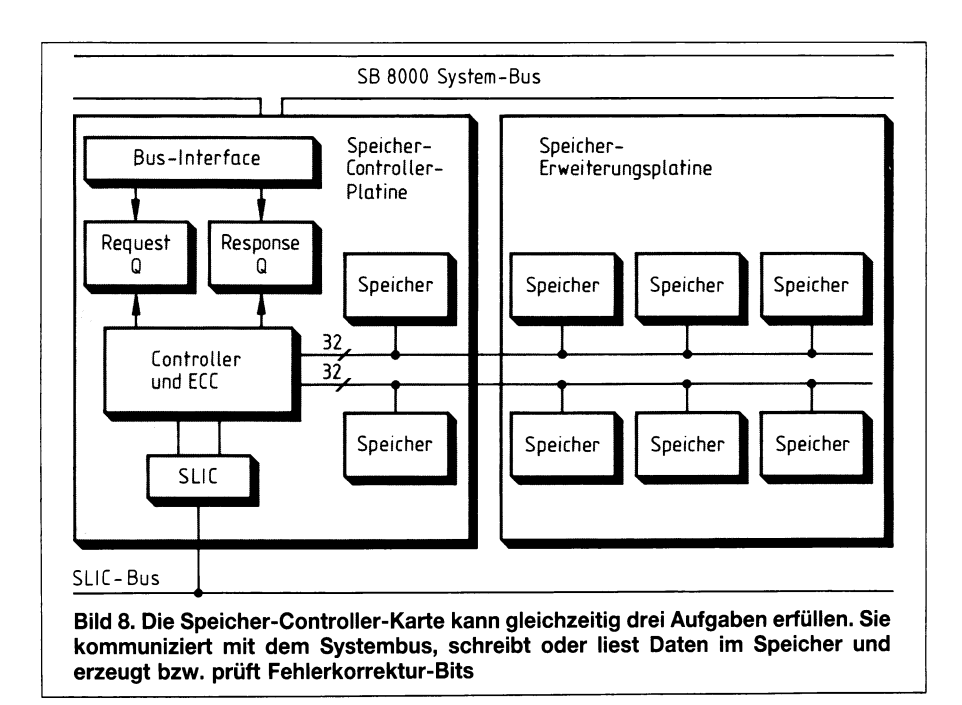


Bild 8. Die Speicher-Controller-Karte kann gleichzeitig drei Aufgaben erfüllen. Sie kommuniziert mit dem Systembus, schreibt oder liest Daten im Speicher und erzeugt bzw. prüft Fehlerkorrektur-Bits

ren, die von der Software entschieden wird. Ein solcher Vorgang besteht typischerweise aus dem Testen der verschiedenen Subsysteme, Abschalten des fehlerhaften Moduls und einem neuen Systemstart.

Globaler Speicher

Bild 8 zeigt das Funktionsdiagramm des Hauptspeichers. Um die sehr große Bandbreite für die SPP-Architektur erreichen zu können, hat der Speicher eine Breite von 64 Bit, so daß sich 8 Byte gleichzeitig übertragen lassen. Der 8-Byte-Transfer wird sehr häufig für das Füllen der Cache-Speicher und E/A-Operationen benutzt. Die Pipeline-Natur der Speicher-Steuerkarte führt dazu, daß drei Funktionen gleichzeitig ausgeführt werden können. Während eine Anfrage, die über den Bus kommt, in die Warteschlange eingefügt wird, greift der Speicher gleichzeitig auf den Speicher zu, um frühere Anfragen zu befriedigen, außerdem werden Fehler-Korrektur-Codes für die Daten erzeugt bzw. geprüft. Ebenfalls zur gleichen Zeit kann die Antwort-Warteschlange validierte Daten früherer Anfragen aussenden. Auf diese Weise kann ein einziger Controller 8 Byte in 300 ns vollständig behandeln.

E/A-Interfaces

Jede E/A-Kanal-Prozessorkarte enthält einen 16-Bit-Mikroprozessor (NS32016). Dessen Befehlssatz ist kompatibel mit der CPU 32032, die sich in den Prozessor-Subsystemen befindet. Verwendet wird er zur Interpretierung der Kanal-Kommandos sowie zum Ansteuern der E/A-Adapter (*Bild 9*). Der Prozessor auf der ersten Kanalkarte verhält sich außerdem wie der Diagnoseprozessor. Seine eigenen ROM/RAM-Bereiche sowie die SLIC-Verbindung zu anderen Modulen stellen sicher, daß diese Einheit auch dann noch richtig arbeitet, wenn irgendwo im System Hardwarefehler auftreten.

Die E/A-Karte enthält auch zwei Standard-Interfaces, nämlich den Ethernet-Anschluß nach IEEE 802.3 sowie die SCSI-Schnittstelle für den Anschluß von Massenspeichern. FIFO-Puffer und DMA-Controller, die für den 8-Byte-Transfer ausgelegt sind, befinden sich auf der Platine, um die Zeitverzögerung zwischen dem Eintreffen der Daten von den Peripherieeinheiten und deren Ablegen im Hauptspeicher zu verhindern. Die hohe Daten-Übertragungsfrequenz kann ein schnelles Dateisystem für UNIX 4.2 bsd unterstützen.

Neben maximal vier Hochgeschwindigkeits-E/A-Kanal-Platinen lassen sich bis zu vier Multibus-Koppler in das System Balance 8000 einfügen. Damit ist es einem Benutzer möglich, eigene Hardware am System anzuschließen oder multibus-kompatible Produkte anderer Hersteller zu verwenden.

Das Betriebssystem

Aus verschiedenen Gründen wählte man die Version 4.2 bsd von UNIX als Betriebssystem für den Computer Balance 8000. UNIX ist ein Mehrbenutzer-Multitasking-Betriebssystem mit einer umfangreichen Bibliothek verknüpfter Dienstprogramme. In den letzten Jahren hat es sich zu einem standardmäßigen Betriebssystem für technische Anwendungen entwickelt. Die Version 4.2 bsd unterstützt nicht nur virtuelle Speicher mit Demand-Paged-Zugriff, so daß mehrere große Programme gleichzeitig sehr effektiv abgearbeitet werden können, sondern sie erlaubt auch wesentlich schnellere Schreib- und Lese-Vorgänge in der Datei über das „Fast File System“.

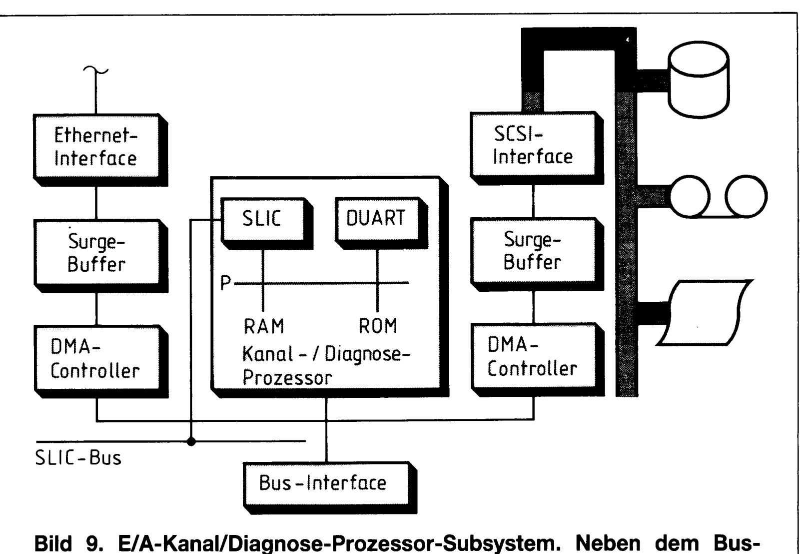


Bild 9. E/A-Kanal/Diagnose-Prozessor-Subsystem. Neben dem Bus-Interface besitzt diese Karte einen Ethernet-Anschluß sowie eine SCSI-Schnittstelle

Außerdem wird die Kommunikation über lokale Netzwerke unterstützt.

Obwohl UNIX für den Benutzer ein Multi-User-Multi-Tasking-System ist, setzt man bei Standard-Implementierungen voraus, daß zu einer bestimmten Zeit immer nur ein Prozessor den Betriebssystem-Code benutzen kann. Die nach außen hin sichtbare konkurrierende Verarbeitung wird mit Hilfe des Zeitmultiplex-Verfahrens erreicht. In echten Multiprozessorsystemen muß der Kern für jeden Prozessor einzeln verfügbar sein. Aus diesen Gründen mußte die Portierung von UNIX auf den Balance 8000, die die Bezeichnung „DYNIX“ trägt, in verschiedenen wichtigen Bereichen verändert werden, z. B. bei der Interrupt-Verteilung, beim Process-Scheduling sowie bei der virtuellen Speicherverwaltung.

(Übersetzung: Be)

Literatur

[1] *Schindler, M.*: Multiprozessor-Architekturen: Höhere Leistung nur mit neuen Konzepten. ELEKTRONIK 1984, H. 17, S. 39...44.

Ingwar Larsson

Supermikros unterstützen Echtzeit-Unix

Die Auswahl einer zukunftssicheren Prozessorfamilie wird für Entwickler von Computersystemen immer wichtiger. Die richtige Wahl führt zu wirtschaftlichen Vorteilen und sichert die Konkurrenzfähigkeit. Das gleiche gilt auch für die System-Software. Es ist außerordentlich wichtig, das richtige Betriebssystem zu wählen. Es sollte auch von vielen anderen Entwicklern benutzt werden. Damit steht eine ausreichende Software-Basis zur Verfügung. Dies wiederum ist erforderlich, um die Computer für Anwender interessant zu machen. An einem praktischen Beispiel wird das in diesem Beitrag gezeigt.

Eine der Firmen, die überzeugt ist, die richtige Wahl unter den Prozessorfamilien getroffen zu haben, ist Dataindustrier AB aus Schweden (DIAB). Dieses Unternehmen entschied sich bei einigen Entwicklungen für die 32000-Familie. Ein Beispiel ist der 16-Bit-Super-Mikrocomputer DS90, mit einem Unix-kompatiblen Echtzeit-Betriebssystem. Ein weiteres Beispiel ist der 32-Bit-Super-Mikrocomputer DS60/32, der mit zwei Prozessoren ausgestattet werden kann. Als drittes Beispiel dient der HR/32, ein schnelles Grafiksystem mit hoher Auflösung.

Warum die 32000er-Familie

Warum wählten die Verantwortlichen von DIAB die 32000-Prozessorfamilie? Dafür gibt es mehrere Gründe:

Erstens sind alle Chips erhältlich, so daß die Entwickler wichtige Funktionen nicht mit anderen Chip-Familien realisieren müssen, wobei dann in der Regel Interface-Probleme auftreten. Damit ergab sich der Vorteil einer sehr kurzen Entwicklungsphase. In Zukunft ist es einfach, auch auf noch folgende Modelle Software, die bereits geschrieben ist, zu portieren.

Ein weiterer Grund ist die Tatsache, daß National Semiconductor eine Strategie verfolgt, nach der die Produkte innerhalb der Familie sowohl aufwärts- als auch abwärtskompatibel sind. Damit halten sich die Entwicklungskosten für die Software in Grenzen. Dies ist besonders wichtig, wenn neue Chips und neue Entwicklungen in immer kürzeren Abständen auf den Markt kommen.

Auch ein wichtiger Grund für die Wahl der 32000er-Familie ist das „saubere" Bus-Interface. Es ermöglicht die schnelle Entwicklung eines neuen Computers, weil es keine Probleme beim Anschluß von Speichern oder anderen Hilfsbausteinen gibt. Zum Beispiel kann die 32000-Familie Speicher mit unterschiedlichen Zugriffszeiten dynamisch verwalten. Wartezyklen werden eingefügt, wenn das erforderlich ist. Die Prozessoren lassen sich außerdem einfach programmieren. Der Befehlssatz ist vollständig orthogonal, d. h. die Adressierungsarten lassen sich in jeder Weise kombinieren. Dies macht die Software kompakt. Bei DIAB hat man herausgefunden, daß Programme, die für die Serie 32000 geschrieben wurden, etwa 30 % weniger Umfang haben als vergleichbare Programme für die 68000-Familie. Nun zu den bei DIAB realisierten Systemen:

Modularer Aufbau

Beim DS90 handelt es sich um einen Mehrbenutzer-Super-Mikrocomputer mit 16-/32-Bit-Struktur mit dem Betriebssystem Unix. Dieses System ist so ausgelegt, daß es mit normalen Terminals, Personal-Computern und anderen Rechnern zusammenarbeiten kann. Einen modularen Aufbau gibt es sowohl bei der Hardware, als auch bei der Software. Die Struktur hat offensichtliche Vorteile und bewährt sich ähnlich wie die Echtzeit-Eigenschaften des Betriebssystems. Darüber hinaus erlaubt der modulare Aufbau einen einfachen und schnellen Service, weil sich einzelne Funktionseinheiten schnell austauschen lassen. Außerdem ergibt sich ein großer Grad an Freiheit und Flexibilität bei der Erweiterung von Hard- und Software.

Die Fließkomma-Hardware

Das System DS90 arbeitet mit dem Prozessor NS32016, der Speicherverwaltungseinheit NS32082

sowie der Fließkomma-Einheit NS32081. Zusammen mit dem Interrupt-Controller sowie einem seriellen Kommunikations-Controller bilden diese Chips die wichtigsten Teile auf der CPU-Platine.

Die Prozessor-Platine ist an einem Datenbus angeschlossen, der die Bezeichnung „DMI-Bus" trägt. Der Hauptspeicher besteht aus Platinen, die jeweils 512 KByte oder 2 MByte Speicherkapazität haben. Das System läßt sich auf insgesamt 8 MByte ausbauen. In der Grundausführung ist es mit 1 MByte Hauptspeicher bestückt. Eine 8-Zoll-Disketteneinheit mit 1 MByte Speicherkapazität erlaubt den Programmaustausch und die Anfertigung von Sicherungskopien. Alle Grundausführungen haben Winchester-Plattenspeicher mit unterschiedlichen Kapazitäten. Einige Modelle lassen sich auch mit Magnetband-Streamer-Laufwerk liefern.

Arbeitsstationen können in einer Sternkonfiguration lokal über V.24-Interfaces angeschlossen werden. Auch Drucker lassen sich über die V.24-Schnittstelle mit dem Rechner verbinden. Jede E/A-Platine hat vier V.24-Kanäle und einen SCSI/SASI-Anschluß für Winchester- und andere externe Massenspeicher-Einheiten. Außerdem sind auf der CPU-Platine zwei V.24-Anschlüsse. Insgesamt stehen damit standardmäßig sechs V.24-Schnittstellen zur Verfügung.

Die sogenannten „Data-Board"-Computer und andere Systeme vom Typ DS90 lassen sich auch über lokale Netzwerke verbinden. Dazu dient entweder das D-Net oder das Ethernet. Alle Grundversionen des DS90 befinden sich in einem Data-Board-E/A-Gestell mit Platz für fünf Platinen, die sich für LAN-Anschluß, externe Kommunikation und Prozeß-Steuerinterfaces nutzen lassen. Damit eignet sich das System DS90 auch für industrielle Anwendungen.

Benchmark-Test

Der Sinn von Benchmark-Tests wird schon seit langem diskutiert, denn es ist umstritten, wie genau sie die wirkliche Leistung eines Computersystems zeigen können. Wenn man allerdings zwei Computersysteme miteinander vergleicht, läßt sich mit Hilfe von Benchmark-Tests doch sehr schön zeigen, was man von einem Computer zu erwarten hat.

Die Zeitschrift „Personal Computer World" benutzt acht Basic-Programme, um Computer zu testen. Als diese Programme auf einem System DS90 mit D-Basic-V abliefen, stellte sich heraus, daß das System DS90 etwa zweimal so schnell wie ein IBM-PC/AT war. Ein weiterer klassischer Benchmark-Test, der sehr häufig von der Zeitschrift „Byte" verwendet wird, ist „Sive of Erostothenes". Dieses Programm findet alle Primzahlen zwischen 1 und 8192. Ein DS90 mit der CPU NS32016, die mit 8 MHz läuft, benötigt dafür 2,8 s. Ein weiterer Benchmark-Test ist „fptest", ein Fließkomma-Programm. Hiermit läßt sich die Fähigkeit zur Verarbeitung von Zahlen messen. „fptest" invertiert eine Matrix und zeigt, wie viele Flops (Floating-Point-Operations per Second) ein Computer ausführen kann. Dieses Programm wird vom DS90 in 3,7 s abgearbeitet. Das entspricht 16,8 KFlops. Im Vergleich dazu braucht das System „Apollo Domain" für dieses Testprogramm 15 s. Wenn man das System DS90 mit Minicomputern von Digital Equipment vergleicht, findet man ebenfalls einige interessante Resultate. Das System DS90 ist schneller als eine VAX-11/750 (etwa 5,5 KFlops), aber langsamer als die VAX-11/780 (etwa 16 KFlops).

Grafikverarbeitung

DS90 läßt sich mit einem Interface für hochauflösende Grafik erweitern. Dieses Interface besteht aus zwei Platinen, die über den Data-Board-Bus des Hauptrechners verbunden sind, wobei das Betriebssystem bereits die Software-Treiber umfaßt, die zur Unterstützung notwendig sind. Die Platinen basieren auf dem Grafik-Prozessor NEC 7220, außerdem ist die Logik sowie der Speicherplatz für folgende Funktionen vorhanden:
- mehrere Display-Seiten;
- zwei Display-Fenster, die sich individuell im Grafikspeicher positionieren und durchlaufen lassen;
- Zooming;
- Blinken;
- vom Benutzer festgelegte Farben (bis zu acht Farben auf dem Bildschirm);
- Zeichengenerator mit ASCII-Zeichen, die sich vom Benutzer neu definieren lassen;
- Symbole und grafische Zeichen;
- Vordergrund-/Hintergrund-Speicherebenen;
- grafischer Speicher mit 512 × 512 Pixeln (optionell 2048 × 512);
- Anpassungsmöglichkeit für verschiedene Monitortypen mit Hilfe von Software-Kommandos.

Auf die Platinen kann man über den Grafik-Handler HR von jeder Sprache aus zugreifen. In D-Basic-V sind entsprechende Ausdrücke bereits vorhanden, die die Platinen direkt ansteuern.

Doppelprozessoren für die Industrie

Die Super-Mikrofamilie DS60 ist in erster Linie für industrielle Anwendungen konzipiert. Es handelt sich um einen „echten" 32-Bit-Computer, der sich mit zwei Mikroprozessoren ausrüsten läßt, die parallel laufen. Es ist allerdings auch möglich, den DS60 mit nur einem Prozessor zu betreiben. Bislang läuft der DS60 mit einem NS32032, später lassen sich auch zwei CPUs vom Typ NS32132 verwenden, wobei ein Baustein als Master und der andere als Slave arbeitet. Beide teilen sich einen Speicher, arbeiten aber unterschiedliche Teile des Programms ab. Damit erhält die Doppelprozessor-Version des DS60 etwa die doppelte Leistung eines Einprozessor-Systems. Diese Doppelprozessor-Ausführung ist ein

gutes Beispiel dafür, wie einfach Systeme mit der 32000-Familie zu entwickeln sind. Es gibt keine Probleme, die CPUs miteinander zu verbinden und die Leistungsdaten zeigen, daß nur wenig Overhead für die Verteilung des Programmablaufs notwendig ist. Hier zeigt sich auch, daß D-NIX für Mehrprozessor-Anwendungen sehr gut geeignet ist.

Integrierte Datenverarbeitung

Das System DS60 stellt für die Firma DIAB die Grundlage zu einem CIM-Konzept (Computer-Integrated-Manufacturing) dar, das die Bezeichnung „ISG 90" trägt. Auf der Fertigungsebene gibt es Computer, die Maschinen über industrielle Controller steuern. Diese Computer sind auf der Grundlage der CPUs NS32016 aufgebaut. Auf der nächsten Ebene gibt es das System DS60 mit einem einzigen NS32032-Prozessor. Dieser Rechner ist zuständig für die Verwaltung und Steuerung der Computer auf der Fertigungsebene. Hier werden auch die Statusreports für die Manager im Fertigungsbereich erzeugt. Auf der obersten Ebene läuft ein DS60-Computer mit zwei Prozessoren vom Typ NS32132. Er produziert Statistiken, Reports und hat die Gesamtverantwortung für alle Computer auf den unteren Ebenen. Die Strategie für ein CIM-Konzept hängt davon ab, wie schnell die Kommunikation zwischen verschiedenen Ebenen erfolgen kann und ob die Möglichkeit zur Echtzeit-Verarbeitung besteht.

System für CAD

Eine weitere interessante Entwicklung mit der Serie 32000 ist das Grafik-System HR/32. Dieses System ist für CAD-Anwendungen und andere Applikationen geeignet, bei denen die Grafikverarbeitung schnell durchgeführt werden muß. Das System HR/32 befindet sich auf zwei Platinen, einem Prozessor-Board und einem Video-Board. Es arbeitet grundsätzlich mit ein und derselben Prozessor-Platine; der Benutzer kann allerdings zwischen zwei verschiedenen Video-Boards wählen, die sich in bezug auf die Auflösung unterscheiden. Die einfachere der beiden Ausführungen hat eine Auflösung von 640 × 512 Pixeln mit 256 Farben. Bei Verwendung von 16 Farben läßt sich die Auflösung auf 1024 × 768 Pixel steigern. Dies bedeutet eine Pixel-Frequenz von etwa 65 MPixel/s. Die zweite Video-Platine bietet eine Auflösung von 1280 × 1024 Pixel, 256 Farben, und die Pixel-Frequenz liegt bei 120 MPixel/s. Die Prozessor-Platine arbeitet auf der Grundlage einer CPU NS32016, die zusammen mit der Fließkomma-Einheit NS32081 und dem DRAM-Controller DP8419 von National Semiconductor und dem CRT-Controller HD63484 (ACRTC) von Hitachi arbeitet. Die Kommunikation mit dem Hostrechner und den Peripherie-Einheiten erfolgt über vier serielle V.24-Ports. Zwei davon dienen zur Kommunikation zwischen dem Hostrechner und einem Terminal (oder einer Tastatur). Die anderen beiden lassen sich zum Anschluß von Hardcopy-Einrichtungen, z. B. Drukker oder Plottern bzw. einem Digitalisierer, einem Track-Ball oder einer „Maus" verwenden.

Verdeckte Linien nicht zu sehen

Das Grafiksystem HR/32 übernimmt einen großen Teil der rechenintensiven Aufgaben, die sonst den Hostrechner belasten. CPU und FPU führen viele Transformationen und andere Berechnungen aus. Während diese abgearbeitet werden, lassen sich Kommandos zum Grafikprozessor senden. Dieser verwaltet die Bildschirm-Darstellung und viele Funktionen, die sonst vom Hostrechner übernommen werden. Die Fähigkeiten des HR/32 kommen insbesondere bei komplexen Grafikaufgaben zur Geltung, z. B. bei der Modell-Simulation. Das System kann beispielsweise verdeckte Linien unterdrücken, Koordinaten-Transformationen und ähnliches vornehmen. Ein weiteres interessantes Merkmal ist die Benutzung des SASI-Interfaces. Dieses Interface läßt sich für zwei verschiedene Zwecke benutzen. Das Grafiksystem kann einerseits damit schnelle Datentransfers vornehmen, andererseits läßt sich diese Schnittstelle für das lokale Speichern von Daten benutzen. Prinzipiell stellt das System HR/32 eine vollständige Unix-Workstation dar. Wird ein lokales Winchester-Laufwerk hinzugefügt und die CPU durch die Kombination von CPU und MMU ersetzt, stellt dieses Grafiksystem eine vollständige Engineering-Workstation mit sehr hoher Verarbeitungsleistung dar.

Unix für Echtzeit-Anwendungen

Jede Hardware ist allerdings nur so gut wie die Software, die darauf läuft. Um die Leistung der Serie 32000 ausnutzen zu können, muß ein gutes Betriebssystem vorhanden sein, das auch Entwicklungswerkzeuge für den Programmierer enthält. Ohne eine solche Unterstützung läßt sich Anwendungs-Software nicht erstellen.

Unix entwickelte sich sehr schnell zum weitverbreiteten Mehrbenutzer-Betriebssystem bei kommerziellen Anwendungen. Im technischen und industriellen Bereich hat es allerdings noch keine große Bedeutung erlangt, weil die meisten Unix-Systeme keine Echtzeit-Verarbeitung beinhalten. Mit D-NIX, einem Unix-kompatiblen Echtzeit-Betriebssystem, das von DIAB entwikkelt wurde, gibt es jetzt ein schnelles Unix-System für kommerzielle Prozeßsteuerungs- und Kommunikations-Anwendungen. Bei D-NIX sind die Unix-Funktionen mit Echtzeit-Verarbeitungsfunktionen kombiniert worden. D-NIX unterstützt darüber hinaus auch virtuelle Speicherverwaltung.

Geschwindigkeit entscheidet

Kritisch bei Echtzeit-Betriebssystemen ist die schnelle Reaktion auf externe Ereignisse. In erster Linie muß die Interrupt-Verarbeitung schnell sein. Auch das Prozeß-Scheduling, das durch externe Ereignisse bestimmt wird, muß mit hoher Geschwindigkeit ausgeführt werden. Um die Zeitverzögerung beim Auftreten eines Ereignisses und dem Scheduling-Prozeß möglichst gering zu halten, hat man das D-NIX-Environment in vier Teile aufgeteilt, die aus vier Arbeitsebenen bestehen. Ebene 1 ist die Kern-Ebene, auf der die Kontext-Umschaltung stattfindet. Auf Ebene 2 werden die Systemdaten manipuliert, aber keine Scheduling-Funktionen ausgeführt. Ebene 3 ist eine System-Erweiterungsebene, auf der die meisten System-Verwaltungsvorgänge stattfinden, zu denen auch das Scheduling gehört. Ebene 4 ist die Benutzer-Ebene. Auf den Ebenen 1 bis 3 läuft der Prozessor im Betriebssystem Supervisor-Mode. Ebene 1 und 2 sind nicht sehr umfangreich, so daß die Zeitverzögerung von einem Ereignis bis zu einer Scheduling-Operation auf ein Minimum reduziert wird.

Trap-Queue

In einer Echtzeit-Umgebung ist es sehr häufig notwendig, auf das Auftreten verschiedener Ereignisse zu warten. Bei D-NIX wird das von einem Systemmechanismus übernommen, der die Bezeichnung „Trap-Queue" trägt. Diese Einrichtung sieht aus wie ein normaler Ein-/Ausgabe-Kanal, über den ein Benutzer gespeicherte Daten freizügig lesen kann. Um diese Funktion benutzen zu können, muß der Benutzer zunächst die Trap-Queue öffnen. Danach kann er Systemaufrufe vornehmen, um entweder zu lesen und zu schreiben oder auf ein externes Ereignis zu warten. Der Benutzer kehrt vom Systemaufruf direkt zurück und nimmt zusätzliche Aufrufe unabhängig von der Beendigung vorhergehender Aufrufe vor. Wenn alle „gleichzeitigen" Systemaufrufe plaziert wurden, gibt der Benutzer ein Lesekommando von der Trap-Queue und wartet, bis einer der Systemaufrufe beendet wurde. Die Daten, die aus der Trap-Queue zu lesen sind, identifizieren den abgeschlossenen Systemaufruf, und der Benutzer kann entsprechende Aktionen veranlassen.

Bit-Mapped-Files

Der File-Handler von D-NIX arbeitet mit Bit-Mapped-Zuweisung im Gegensatz zur freien Liste, die bei Unix zur Anwendung kommt. Das Bit-Mapping ermöglicht die Identifizierung großer, kontinuierlicher Speicherblöcke für die Zuweisung kontinuierlicher Dateien. Dies ist sehr nützlich bei Echtzeit-Anwendungen, bei denen große Datenmengen auftreten.

MS-DOS-Verarbeitung

Ein weiterer sehr leistungsfähiger Mechanismus in D-NIX ist der Handler, der eine Erweiterung der Unix-Architektur darstellt. Ein Handler ist ein Service-Prozeß, der wie alle anderen ähnlichen Prozesse Hilfsfunktionen für andere Prozesse übernimmt. Der Handler ist im Betriebssystem untergebracht, wenn er auf einer Datei oder einer Directory im File-System aufgesetzt ist. Alle Systemaufrufe, die sich auf die Datei oder Directory beziehen, werden daher direkt an den aufgesetzten Handler geleitet und nicht zum File-Handler. Der Mechanismus ermöglicht es, File-Handler für andere Dateisysteme einzurichten, die das ursprüngliche D-NIX-Dateisystem ergänzen. Derzeit sind File-Handler für MS-DOS und CP/M verfügbar. Bei Anwendungen gibt es keinen Unterschied zwischen dem Schreiben einer D-NIX-File oder einer MS-DOS-File auf einer Floppy-Disk, wenn man über einen aufgesetzten MS-DOS-File-Handler geht.

Es ist einfach möglich, einen File-Handler zu schreiben, wenn das gesamte Dateisystem im Speicher geladen ist. Solch ein File-System ist sehr schnell und deshalb insbesondere für Echtzeit-Prozesse geeignet. Im Falle, daß der Speicher nicht groß genug für das gesamte Dateisystem ist, benutzt der virtuelle Speichermechanismus automatisch Plattenkapazität in einer schnellen und transparenten Betriebsart.

Der Betrieb in einem lokalen Netzwerk wird ebenfalls durch Handler-Funktionen implementiert. Netzwerk-Handler, die sich in verschiedenen Maschinen befinden, können miteinander transparent kommunizieren. Daher gibt es bei einer Applikation keinen Unterschied zwischen Zugriffen auf eine Datei, die sich auf der Maschine befindet, auf der die Anwendungs-Software gerade läuft, und einer Datei auf einer anderen angeschlossenen Maschine. Der größte Teil des Systems ist in Form von Benutzer-Prozessen implementiert, wobei lediglich der Netzwerktreiber im Betriebssystem untergebracht ist. Datenbanken und andere System-Ressourcen sind natürlich in ähnlicher Weise wie Handler zu implementieren.

Kompatibilität zum System V

D-NIX ist vollständig kompatibel mit Unix System V. Damit erfüllt es alle Anforderungen, die an die Interface-Definition auf Basisebene und Kern-Erweiterungsebene gestellt werden. Heute läßt sich D-NIX auf allen Computern implementieren, die auf der Basis der NS32000- oder der 68000-Familie aufgebaut sind.

Unix mit Fenstern

Der Window-Handler, der in C geschrieben ist, läuft unter D-NIX und erfordert ein Bit-Mapped-Display, z. B. HR/32. Bis zu 16 Fenster lassen sich z. B. als „Termi-

nals", Menüs usw. verwenden. Ein „Terminal"-Fenster verhält sich wie ein normales Terminal, auf dem Standard-Unix-Programme laufen können. Mit Hilfe einer „Maus" kann man beispielsweise das Fenster beliebig verschieben, die Abmessungen verändern oder eine Scroll-Funktion steuern. Jedes Fenster emuliert ein erweitertes VT-100-Terminal, wobei auch Grafikfunktionen unterstützt werden, z. B.:

- Zeichnen von Linien mit Hilfe von Mustern;
- Zeichnen von Kreisbögen mit Hilfe von Mustern;
- Ausfüllen von Rechtecken mit Hilfe von Mustern;
- Einfärben mit Hilfe von Mustern;
- Kopieren.

Icons werden ebenfalls unterstützt. Sie lassen sich zum Starten von Programmen oder zur Auswahl innerhalb eines Programmes benutzen, indem man mit der „Maus" an die entsprechende Stelle fährt und die Taste betätigt. In den verschiedenen Fenstern kann der Anwender unterschiedliche Zeichen-Fonts benutzen, die in Dateien abgelegt sind. Im gleichen Fenster kann man auch zwischen verschiedenen Fonts umschalten. Die Zoom-Funktion ist zeichenweise möglich, so daß sich in einem Fenster die Zeichen vergrößern lassen. Programme kommunizieren mit dem Window-Handler über das Betriebssystem. Es besteht die Möglichkeit, Fenster von Programmen in C, Pascal, Fortran und D-Basic-V aus zu öffnen und zu manipulieren.

Entwicklungswerkzeuge

Der Window-Handler ist in C geschrieben, wie der größte Teil der Software für DS90 und jeden anderen Computer, der unter Unix läuft. Es existieren mehrere Werkzeuge für die Software-Entwicklung, in erster Linie alle die üblichen Tools aus dem Unix-Environment. Alle Programme, die in C unter Unix oder Xenix geschrieben sind, laufen auch unter D-NIX.

Es gibt auch einige Entwicklungswerkzeuge von DIAB. Auf der untersten Ebene ist dies der 32000-Assembler und -Linker. Auf der nächsthöheren Ebene gibt es einen C-Compiler und ein semicompilierendes Basic, das D-Basic-V. Ein interessantes Merkmal von D-Basic ist die Tatsache, daß diese Sprache ein Interface zum relationalen Datenbank-Manager „MIMER" besitzt. D-Basic enthält Kommandos, mit deren Hilfe sich Datenbanken erzeugen, modifizieren und pflegen lassen. Damit ist D-Basic eine gute Entwicklungssprache für kommerzielle Software. Der Software-Entwickler kann die schwierige Datenbankverwaltung „MIMER" überlassen.

Für weitere traditionelle Software-Entwicklungsaufgaben stehen „LIB-C", eine C-Library mit vielen nützlichen Routinen, und „SiV", ein Full-Screen-Editor für Programmierer zur Verfügung. Es gibt außerdem verschiedene Handler, die sich in anderen Programmen unterbringen lassen. Ein Z80-Emulator ist ebenfalls erhältlich.

Leichter Einstieg mit Starter-Kits

Der NS32016-Starter-Kit und der NS32032-Starter-Kit sind kostengünstige Entwicklungs- und Evaluationswerkzeuge, die den Einstieg in die 32-Bit-Prozessortechnik ermöglichen, ohne daß hierfür ein großer Investitionsaufwand notwendig ist. Sie bestehen aus den Boardcomputern DB32016 bzw. DB32000, dem TDS (Tiny Development System) – einer EPROM-residenten Firmware mit Assembler und Editor – sowie einer umfangreichen Dokumentation. Außerdem enthalten die Starter-Kits das Lehrprogramm von Nationals europäischem Trainingszentrum in München mit Hinweisen auf Möglichkeiten für Kurse im Hause des Kunden, wenn mehrere Ingenieure einer Firma mit der Serie 32000 vertraut gemacht werden sollen.
Im Lieferumfang des NS32032-Starter-Kits – als Option auch für den NS32016-Starter-Kit verfügbar – ist darüber hinaus ein TOOLKIT enthalten. Das TOOLKIT umfaßt die Monitor-Firmware, Dokumentation, Handbücher und Evaluations-Cross-Software der Firma Owl-Computer Inc. für den IBM-PC unter MS-DOS. PC-Compiler für Pascal und „C" sind ebenfalls von unabhängigen Software-Häusern lieferbar.
Kern des NS32016-Starter-Kits ist das vollständig Multibus-kompatible Multi-Master-Entwicklungsboard DB32016. Der Kit enthält ferner die gesamte Dokumentation und die Software, um mit den CPUs, den Slave-Prozessoren und Peripherie-Bausteinen der Serie 32000 zu arbeiten, Programme zu entwickeln und auszutesten.
Wesentlicher Bestandteil des NS32032-Starter-Kits ist das Entwicklungsboard DB32000 mit der CPU NS32032 (oder auch NS32016 oder NS32008). Die Taktfrequenz der CPU beträgt 10 MHz. Neben dem kompletten Chipsatz mit FPU, MMU, ICU und TCU hat die Platine einen 256-KBit-DRAM (erweiterbar bis 1 MB), zwei RS-232-Schnittstellen, bis zu 256 KB PROM-Kapazität sowie ein Wire-Wrap-Feld für kundenspezifische Beschaltungen. Damit auch ein Doppel-Prozeß-Betrieb möglich ist, gibt es auf der Platine einen Sockelplatz zum Einstecken einer zweiten NS32032-CPU: Die Software bildet den zweiten Hauptbestandteil des NS32032-Starter-Kits. Sie besteht aus der TDS-Firmware (EPROM-resident) und dem TOOLKIT-Paket mit der Cross-Software für den IBM-PC. Als Optionen sind für den NS32032-Starter-Kit Pascal- und „C"-Compiler von Owl-Computer verfügbar. Diese Compiler sind allerdings nicht so hoch optimiert und effizient wie die von National Semiconductor entwickelten Compiler. Darüber hinaus verwendet Owl-Computer ein anderes File-Format, so daß eine Übertragung der Programme auf professionelle Entwicklungswerkzeuge, wie die Systeme SYS32, VR32 oder VAX, nicht ohne weiteres möglich ist.
Weiterhin sind für beide Starter-Kits optionell lieferbar:

- das Echtzeit-Betriebssystem EXEC II
- das TDS-Quellenprogramm
- das FPS-Quellenprogramm (Emulation der NS32081-FPU)
- Software-Katalog der unabhängigen Lieferanten (ISVC).

Uwe Krause

Anschluß von 32000-CPUs an den Multibus

Eines der Hauptelemente in einem Computersystem ist der Systembus, der alle Hardware-Elemente miteinander verbindet. Der Bus enthält die erforderlichen Signale zum Verkehr der Hardware-Komponenten untereinander. Speicher- und E/A-Zugriffe, System-Interrupts usw. können über den Systembus ablaufen. Dem Entwickler steht eine Vielzahl von Alternativen zur Verfügung, wenn er ein Interface zu einem bestimmten Bus entwerfen soll. Für eine CPU mit geringer Leistungsfähigkeit mag die Leistung der Schnittstelle von geringerer Bedeutung sein. In diesem Fall kann die Verwendung eines Standard-LSI-Bauteils, das die Interface-Funktionen enthält, empfehlenswert sein. Wenn dagegen eine Hochleistungs-CPU eingesetzt wird und die Leistung des Interface für die Leistung des Gesamtsystems bestimmend ist, kann die bessere Lösung darin bestehen, eine spezielle Schaltung unter Verwendung von TTL-Logik und in Verbindung mit schnellen PALs oder Gate-Arrays zu entwickeln. Dieser Anwendungsbericht zeigt eine praktische Realisierungsmöglichkeit des Arbitrations-Teiles einer Multibus-Interface-Schaltung, die speziell für die CPUs der 32000-Familie entworfen worden ist.

Schaltung

Die hier beschriebene Multibus-Arbitrations-Schaltung verwendet ein schnelles PAL und einige Standard-TTL-Logik zur Realisierung der Arbiter-Zustandsmaschine und anderer Steuerungsfunktionen. Diese Lösung bringt, im Vergleich mit einer auf LSI-Bausteinen basierenden Lösung, neben Geschwindigkeitsvorteilen auch ein höheres Maß an Flexibilität. Eine ausführliche Schaltung dieser Arbitrations-Einrichtung und das Arbiter-Zustandsdiagramm sind in *Bild 1* und *2* dargestellt. *Bild 3* zeigt ein Zeitdiagramm einer Bus-Anforderungs- und Freigabe-Sequenz. In dem gezeigten Fall hatte ein Master mit höherer Priorität die Kontrolle über den Bus, und ein Master mit niederer Priorität forderte während eines Bus-Transfer-Zyklus des laufenden Masters den Bus an.

Wie das Zustandsdiagramm zeigt, verwendet der Arbiter eine Nichtfreigabe-Philosophie, „Parking" genannt. In diesem Schema wird ein einmal belegter Bus am Ende des Zugriffs-Zyklus nicht freigegeben, es sei denn, das Signal zum Erzwingen der Freigabe „FREL" ist low, oder von einem anderen Master ist eine Bus-Anforderung eingetroffen. Letztere Bedingung wird erkannt durch Überwachung der Bussignale BPRN und CBRQ. Eine Bus-Freigabe-Sequenz wird gestartet, sobald CBRQ low ist oder BPRN auf high geht. Der Arbiter geht in Zustand S3 und setzt das Signal „Freigabe der Zyklus-Ende-Erkennung" ECDEC, um die entsprechende Logik das Ende des laufenden Multibus-Transfer-Zyklus der CPU erkennen zu lassen. Wenn die Zyklus-Ende-Bedingung erkannt ist, werden weitere Multibus-Zugriffe der CPU auf dem Board verhindert. Das Signal ECYC wird gesetzt, um dem Arbiter mitzuteilen, daß der Bus freigegeben werden kann. Man beachte, daß beim Ablauf einer verriegelten Operation die Zyklus-Ende-Bedingung erst am Ende des letzten Zyklus der verriegelten Operation und nicht am Ende des laufenden Zyklus auftritt.

Der Verriegelungsmechanismus für die Busfreigabe ist notwendig, um eine saubere Funktion der Arbitrations-Schaltung sicherzustellen, und zwar unabhängig von Bus- und CPU-Taktfrequenz. Die Nichtfreigabe-Philosophie ist insofern sehr wirkungsvoll, als sie für den laufenden Master den Overhead durch Bus-Wechsel erspart, wenn kein anderer Master den Bus verlangt. Das erhöht auch die Zuverlässigkeit, da die Anzahl der für die Bus-Arbitration notwendigen Synchronisations-Operationen minimiert wird.

In dieser Anwendung wurde besondere Sorgfalt für die Bus-Verriegelung verwandt. Dies ist wichtig, vor allem wenn ein Dual-Port-Speicher eingesetzt wird, da Hardware-Deadlocks entstehen können, wenn die verriegelten Zyklen nicht ordentlich abgehandelt werden.

Betrachten wir den Fall von verriegelten Operationen, die Zyklen mit Mehrfachzugriff enthalten, wobei der erste Zugriff auf den Dual-Port-Speicher und einige oder alle der folgenden Zugriffe auf den Systemspeicher gehen. Das kann z. B. vorkommen, wenn die MMU-

Zyklen verriegelt sind, die Seitentabelle der ersten Ebene im Dual-Port-Speicher liegt und einige gemeinsam benutzte Seitentabellen der zweiten Ebene entweder im Systemspeicher oder im Dual-Port-Speicher eines anderen Masters stehen. In diesem Falle würde ein „Hardware-Deadlock" entstehen, wenn die Logik des Dual-Port-Speichers einen verriegelten Zugriff auf die MMU zuteilt (und so einen Verriegelungsmechanismus anstößt, um bis zum Ende der verriegelten Operation Zugriffe von externen Mastern zu verhindern) und in der Zwischenzeit der Bus von einem externen Master verlangt wird, der versucht, auf den Dual-Port-Speicher zuzugreifen. Der Grund ist, daß der Dual-Port-Speicher nicht entriegelt wird, bevor die MMU auf den Systemspeicher zugreift, um die Operation abzuschließen, wäh-

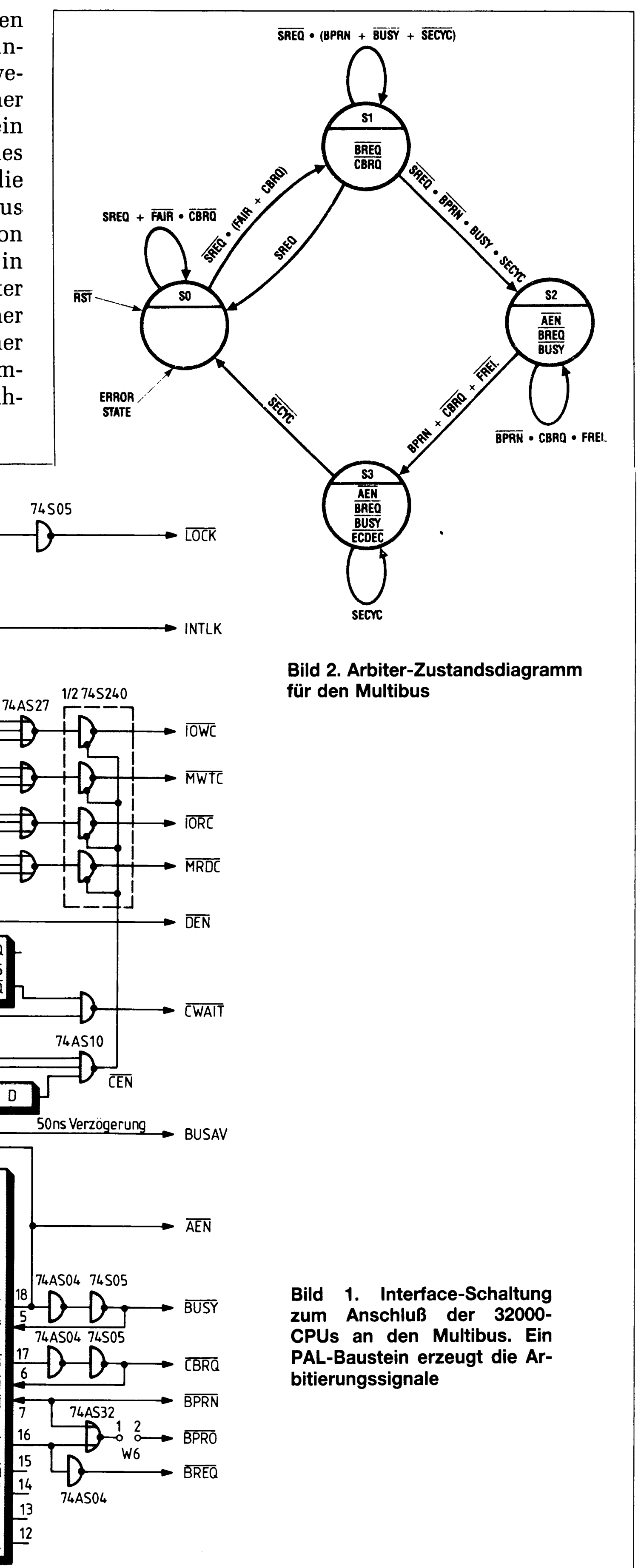


Bild 2. Arbiter-Zustandsdiagramm für den Multibus

Bild 1. Interface-Schaltung zum Anschluß der 32000-CPUs an den Multibus. Ein PAL-Baustein erzeugt die Arbitierungssignale

rend der Multibus vom externen Master nicht freigegeben wird, bevor der Zugriff auf den Dual-Port-Speicher zugeteilt wurde.

Dieses Problem kann dadurch gelöst werden, daß der Bus zu Beginn der verriegelten Operation angefordert wird, selbst wenn der verriegelte Zyklus nicht den Systemspeicher anspricht, und der erste verriegelte Zugriff auf den Dual-Port-Speicher solange verzögert wird, bis der Bus zugeteilt worden ist.

Ein weiterer wichtiger Punkt ist die Erzeugung der Multibus-Read- und Write-Signale. Diese werden von den TCU-Signalen RD und WR abgeleitet. Die steigenden Flanken dieser Signale werden verzögert, um die Timing-Anforderungen des Multibus zu erfüllen. Die Multibus-Read-Signale MRTC und IORC werden verzögert, bis das DBE-Signal gesetzt wird, da nur die Anforderung an die Setup-Zeit der Adressen erfüllt werden muß. Dafür wird für die Multibus-Write-Signale MWTC und IOWC eine größere Verzögerung benötigt, um die Anforderungen an die Setup-Zeit der Daten zu erfüllen. Man beachte, daß die Verzögerungsleitung diesen Zweck nur während des ersten Buszugriffes nach einer Arbitrations-Sequenz erfüllt.

Signale

Einzelheiten über die meisten der in der Multibus-Arbitrationsschaltung verwendeten Signale können in den entsprechenden Multibus-Handbüchern und den Datenblättern der 32000-Familie nachgelesen werden. Es folgt eine Beschreibung derjenigen Signale, die in diesen Dokumenten nicht vorkommen.

Bus Override (OVERR): Dieses Signal kommt aus einem Override-Flipflop und kann benutzt werden, um den Bus während Burst-Transfers festzuhalten. Die maximale Dauer, während der das Signal gesetzt sein darf, darf die kürzeste Time-out-Zeit im System nicht überschreiten. Man beachte, daß das Signal „LOCK" nicht gesetzt wird, während der Bus durch das Signal OVERR überlagert wird, und zwar weil das LOCK-Signal nur für maximal 12 µs gesetzt werden kann, während der Bus, je nach den Systemanforderungen, für mehrere Millisekunden überlagert werden kann.

IO Select (IOSL): Dieses Signal wird vom Adreßdecoder generiert und wird gesetzt, wenn auf den Multibus-I/O-Bereich zugegriffen wird.

Multibus Select (MULSL): MULSL kommt aus dem

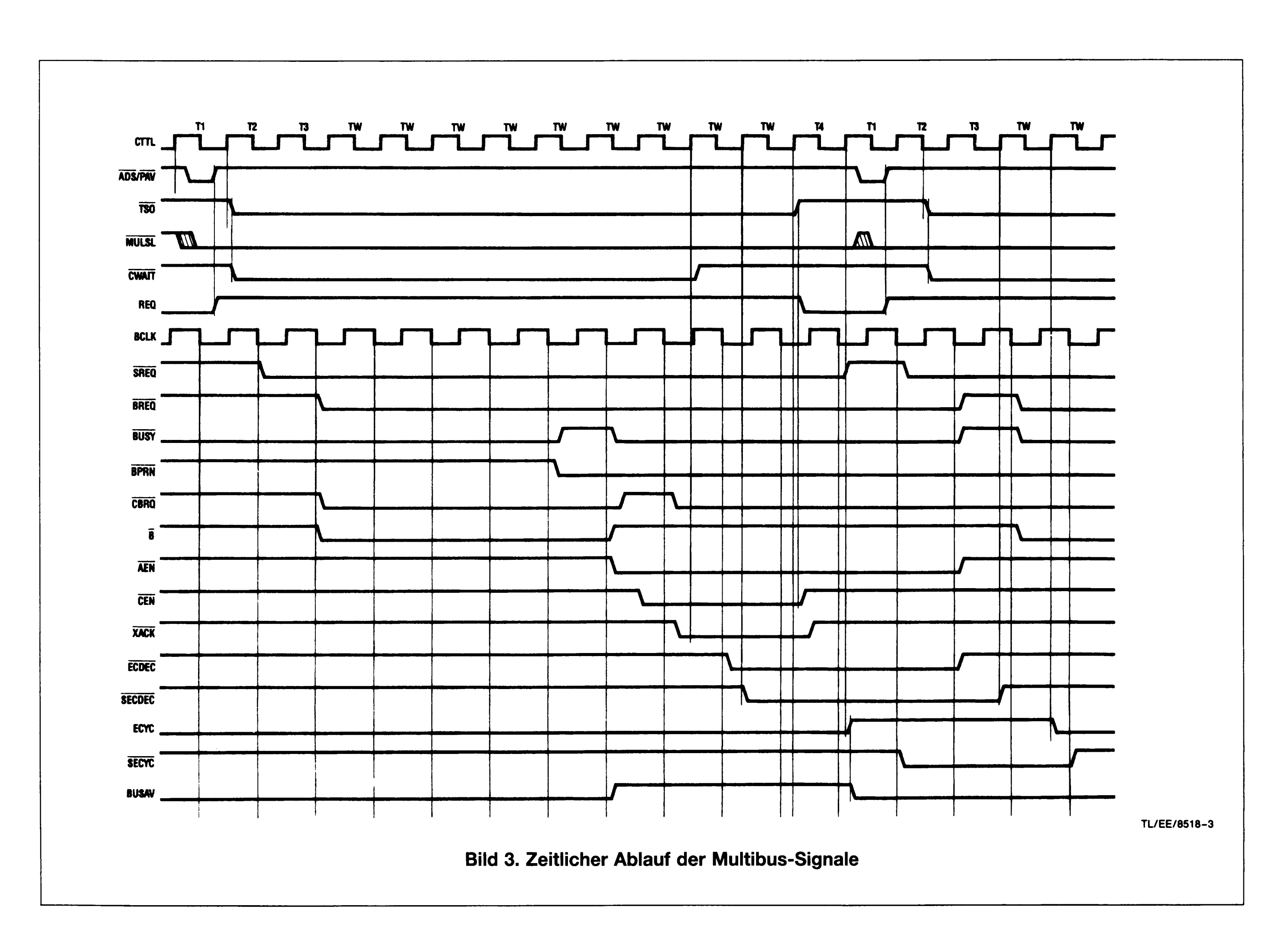


Bild 3. Zeitlicher Ablauf der Multibus-Signale

Adreßdecoder und wird gesetzt, wenn ein Multibus-Speicher- oder I/O-Zugriff verlangt wird.

Time-out (TOUT): Dieses Signal wird von einem ausfallsicheren Timer erzeugt und wird benutzt, um den Zyklus abzubrechen, falls eine Buszuteilung, ein Peripherie- oder Speicher-Acknowledge nicht innerhalb einer bestimmten Zeit kommt.

Address Buffers Enable (AEN): Wenn AEN gesetzt ist, sind die Multibus-Adreßbuffer aktiv.

Data Buffers Enable (DEN): Wenn DEN gesetzt ist, sind die Multibus-Datenpuffer aktiv.

Inerlocked Cycle (INTLK): Aktiv high. Dieses Signal zeigt, wenn es gesetzt ist, an, daß ein verriegelter Zyklus entweder gestartet wird oder läuft. INTLK wird von der Logik des Dual-Port-Speichers verwendet, um verriegelte Zugriffe zu steuern.

Bus Available (BUSAV): Dieses Signal teilt der Logik des Dual-Port-Speichers mit, daß der Bus zugeteilt wurde und ein verriegelter Zugriff gestattet werden kann. Dies ist erforderlich, um Hardware-Deadlocks zu verhindern.

Arbiter-Logik-Gleichungen

$\mathrm{S0} := \mathrm{S0} \cdot (\mathrm{SERQ} + \overline{\mathrm{FAIR}} \cdot \overline{\mathrm{DBRQ}}) + \mathrm{S1} \cdot \mathrm{SREQ} + \mathrm{S3} \cdot \overline{\mathrm{SECYC}} + \overline{\mathrm{RST}} + \mathrm{ERRST}$

$\mathrm{S1} := (\mathrm{S0} \cdot \overline{\mathrm{SREQ}} \cdot (\mathrm{FAIR} + \mathrm{CBRQ}) + \mathrm{S1} \cdot \overline{\mathrm{SREQ}} \cdot (\mathrm{BPRN} + \overline{\mathrm{BUSY}} + \overline{\mathrm{SECYC}})) \cdot \mathrm{RST}$

$\mathrm{S2} := (\mathrm{S1} \cdot \overline{\mathrm{SREQ}} \cdot \overline{\mathrm{BPRN}} \cdot \mathrm{BUSY} \cdot \mathrm{SECYC} + \mathrm{S2} \cdot \overline{\mathrm{BPRN}} \cdot \mathrm{CBRQ} \cdot \mathrm{FREL}) \cdot \mathrm{RST}$

$\mathrm{S3} := (\mathrm{S2} \cdot (\mathrm{BPRN} + \overline{\mathrm{CBRQ}} + \overline{\mathrm{FREL}}) + \mathrm{S3} \cdot \mathrm{SECYC}) \cdot \mathrm{RST}$

Zustandscodierung

$\mathrm{S0} = \mathrm{A} \bullet \mathrm{B} \bullet \overline{\mathrm{C}} \bullet \mathrm{D}$

$\mathrm{S1} = \mathrm{A} \bullet \overline{\mathrm{B}} \bullet \mathrm{C} \bullet \mathrm{D}$

$\mathrm{S2} = \overline{\mathrm{A}} \bullet \mathrm{B} \bullet \mathrm{C} \bullet \mathrm{D}$

$\mathrm{S3} = \overline{\mathrm{A}} \bullet \mathrm{B} \bullet \mathrm{C} \bullet \overline{\mathrm{D}}$

Fehlerzustand

$\mathrm{ERRST} = \mathrm{A} \bullet \mathrm{B} \bullet \mathrm{C} \bullet \mathrm{D}$

Bild 4. Logische Gleichungen für die Arbiterfunktion und die Zustandsdecodierung

```
PAL16R8A
PART #
MULTIBUS ARBITER
NATIONAL SEMICONDUCTOR
CLK REQ RST ECYC BUSY CBRQ BPRN FREL FAIR GND
EN NC NC SECYC C B A D VCC
/A := A · /B · C · D · /SREQ · /BPRN · BUSY · SECYC · RST + /A · B · C · D · RST
    + /A · B · C · /D · SECYC · RST
/B := A · B · /C · D · /SREQ · FAIR · RST + A · B · /C · D · /SREQ · CBRQ · RST
    + A · /B · C · D · /SREQ · BPRN · RST + A · /B · C · D · /SREQ · /BUSY · RST
    + A · /B · C · D · /SREQ · /SECYC · RST
/C := A · B · /C · D · SREQ + A · B · /C · D · /FAIR · /CBRQ + A · /B · C · D · SREQ
    + /A · B · C · /D · /SECYC + /RST + A · B · C · D
/D := /A · B · C · D · BPRN · RST + /A · B · C · D · /CBRQ · RST
    + /A · B · C · D · /FREL · RST + /A · B · C · /D · SECYC · RST
/SREQ := REQ
/SECYC := ECYC
```

Bild 5. PAL-Gleichungen im „PALASM"-Format

Konfiguration

Im folgenden wird gezeigt, wie die verschiedenen in der Multibus-Arbitrationsschaltung enthaltenen Brükken zu verwenden sind.

W1: Diese Brücke ist zu setzen, falls MMU-Zyklen verriegelt werden müssen. Dies kann erforderlich sein, wenn einige der Seitentabellen von verschiedenen Masters gemeinsam benutzt werden.

W2: W2 sollte gesetzt werden, wenn W1 gesetzt ist und gemeinsam benutzte Seitentabellen im Dual-Port-Speicher eines anderen Masters stehen.

W3: Wenn diese Brücke nicht gesetzt ist, erzeugt ein verriegelter Zugriff eine Multibus-Anforderung. Das kann erforderlich sein, um Deadlocks zu verhindern. Diese Brücke kann gesetzt werden, falls alle Zyklen einer verriegelten Operation immer an den Multibus oder den Dual-Port-Speicher gehen. Das könnte die Leistung des Systems dadurch verbessern, daß der Bus nicht mehr zugeteilt werden muß, wenn eine verriegelte Operation nur auf den Dual-Port-Speicher zugreift. Diese Brücke sollte bei Verwendung der jetzigen Versionen der 32000-CPUs nicht gesetzt werden, weil diese CPUs zwischen den Lese- und Schreibzyklen einer verriegelten Operation einen Prefetch ausführen können.

W4: Wenn diese Brücke gesetzt ist, wird der Bus am Ende jedes Zyklus freigegeben. Dies könnte vermieden werden, wenn alle Master im System CBRQ unterstützen, da hieraus eine beträchtliche Leistungsverminderung resultiert. Falls aber ein Master mit niedriger Priorität CBRQ nicht unterstützt, muß W4 gesetzt werden, sonst könnte der Master mit geringer Priorität (der CBRQ nicht unterstützt) den Bus nicht bekommen.

W5: Wenn W5 gesetzt ist, wird ein „Fairneß-Arbitrationsmechanismus aktiviert. In diesem Falle wird eine neue Busanforderung nur dann abgesetzt, wenn alle anderen externen Anforderungen bedingt sind. Damit kann ein serielles Arbitrations-Schema mit drei den Bus stark belastenden Mastern (oder mehr, wenn die Taktfrequenz reduziert wird) eingesetzt werden, und zwar ohne die Gefahr, daß der Bus durch die beiden Master mit der höchsten Priorität monopolisiert wird.

W6: W6 muß gesetzt werden, wenn ein serielles (Daisy Chain) Arbitrations-Schema verwendet wird. Sie muß bei Verwendung eines parallelen Schemas entfernt werden.

Dr. Werner Trattnig

Mikroprozessor-Architektur ohne enge Grenzen:

Weiterentwicklung der 32000-Serie

Eines der nützlichsten Merkmale der 32000-Mikroprozessorfamilie ist die große Bandbreite der Preis-/Leistungs-Optionen, verbunden mit der durchgängig gemeinsamen Architektur als Basis für alle Prozessoren. So reichen zum Beispiel die derzeit verfügbaren Bausteine vom preiswerten Prozessor NS32008, der auf Konsumanwendungen zielt, bis zur CPU NS32332, die mehr Leistung als jeder andere derzeit verfügbare 32-Bit-Mikroprozessor bietet. Im folgenden wird der Prozessor NS32532 vorgestellt, der 1987 verfügbar sein wird und mit einer Leistung von 8...10 MIPS die 32000-Familie nach oben hin erweitert.

Aufstieg innerhalb der Familie

In der Vergangenheit mußten die Systementwickler von 8-Bit- auf 16-Bit- und dann auf 32-Bit-Bausteine umsteigen, um eine substantielle Leistungssteigerung des Systems zu erreichen. Dieser Zwang besteht nun nicht mehr, abgesehen von wenigen hochspezialisierten Applikationen. Ein 32-Bit-Prozessor bietet das ideale Format für die meisten Computeranwendungen. Der Markt wird natürlich weiterhin einen steigenden Bedarf an höheren Leistungen haben, aber dieser Bedarf läßt sich im Rahmen der etablierten 32-Bit-Architekturen decken. Im Hinblick darauf konzentrieren sich alle Entwicklungsziele beim Prozessor-32532 auf den im Mikroprozessor-Markt am meisten umstrittenen Punkt: der Frage nach der Kompatibilität zwischen unterschiedlichen Mikroprozessoren.

Die Kompatibilität wird im allgemeinen in bezug auf die mögliche Weiterentwicklung eines Systems diskutiert. Sie soll einen Weg bieten, auf dem der OEM-Kunde sein vorhandenes System weiter ausbauen kann, um eine höhere Leistungsfähigkeit zu erreichen, ohne dafür die Kosten einer völligen Neuentwicklung in Kauf nehmen zu müssen. Es gibt jedoch auch einen Bedarf für eine genau umgekehrt laufende Fortentwicklung. Auch die gründlichsten Marktuntersuchungen decken nur selten alle Marktmöglichkeiten auf, und es ist deshalb nicht unüblich, daß OEM-Lieferanten einen Bedarf für Produkte mit anderen Preis-/Leistungsmerkmalen entdecken, wenn der Verkauf eines neuen Produktes schon begonnen hat. Im Idealfalle sollten sie in der Lage sein, auf die geänderten Marktverhältnisse zu reagieren, ohne zusätzlich in neue Software investieren zu müssen.

Die Entwicklung von Bausteinen wie den Typen 68020 oder 80386 war sehr stark von dem Wunsch beeinflußt, einen gewissen Grad von Kompatibilität zwischen neuen Bausteinen und deren Vorgängern zu bewahren. Diese Bausteine sind deshalb ihren Vorgängern in vielerlei Hinsicht ähnlich, sie haben jedoch neue Register und Befehle, die dem Bedarf nach gesteigerter Leistung entgegenkommen. Der Nachteil dieser Lösung ist allerdings, daß weder eine wirkliche Aufwärts- noch eine ebensolche Abwärtskompatibilität geboten wird. So kann zum Beispiel ein für den 80386 geschriebener Code dann nicht auf einem 80286 laufen, falls er irgendeines der neuen Register oder einen der neuen Befehle nutzt, die den 80386 zu einem leistungsfähigeren Baustein machen. In der gleichen Weise ergibt sich keine wesentliche Leistungssteigerung, wenn ein Programm für die CPU 80286 auf dem 80386 läuft, sofern der Code nicht überarbeitet wurde, um die neuen Register und Befehle des 80386 zu nutzen.

Keine Kompatibilitätseinschränkungen

Innerhalb der 3200-Familie gibt es keinerlei Einschränkungen der Kompatibilität. Es ist die einzige 32-Bit-Prozessorfamilie, die eine uneingeschränkte Auf- und Abwärtskompatibilität bietet. Wie *Tabelle 1* zeigt, achtete man bei der Entwicklung des 32532 darauf, diesen Vorteil zu bewahren, während dieser CPU-Baustein eine höhere Leistung, einen höheren Integrationsgrad, verbesserte Echtzeit-Unterstützung und die Möglichkeit bietet, zukünftige Fortschritte der Technik zu nutzen. Die Verbesserung der Leistungsfähigkeit beruht zum Teil auf dem Einsatz des 1,5-μm-Doppel-Metall-CMOS-Prozesses und zum anderen Teil auf einigen wichtigen Änderungen in der Implementierung, mit dem Ziel, die Möglichkeiten des neuen Halbleiter-Prozesses voll zu nutzen. In der Einführungsphase werden die Bausteine mit 20 MHz Taktfrequenz betrieben, selektierte Versionen sogar mit 25 MHz. Später, wenn die Einführungsphase für den 1,25-μm-Prozeß beendet ist und dieser zum Standard wird, sind Taktfrequenzen von 25 bzw. 30 MHz möglich.

Bild 1 zeigt eine Blockschaltung der neuen CPU. Man kann daraus ersehen, daß es einige Unterschiede zwischen diesem Typ und früheren CPUs der Serie 32000 gibt. Diese Unterschiede beinhalten „On-Chip-Befehle", „Data-Caches", eine integrierte Speicherverwaltungs-Einheit (MMU = Memory Management Unit) und eine erweiterte Bus-Interface-Einheit (BIU = Bus Interface Unit). Die vier Funktionsblöcke auf der linken Seite des Blockschaltbildes stellen den Befehlssatz-Prozessor dar. In der Mitte sind die MMU und die On-Chip-Buffer zu sehen, die den parallelen Datenfluß vom Speicher zur MMU aufrecht erhalten, so daß der Prozessor nahe an seiner theoretischen Durchsatzkapazität arbeitet. Auf der rechten Seite sitzt die BIU, die die Schnittstelle zum restlichen Teil des Systems bildet. Die wesentlichen Änderungen in diesem Bereich bestehen aus separaten Adreß- und Datenbussen.

Tabelle 1. Entwicklungsziele für die CPU NS32532

- Kompatibilität zur Architektur der Serie 32000
- Höhere Verarbeitungsleistung
- Verbesserte Echtzeit-Unterstützung
- Nutzung zukünftiger Techniken

Beginnend in der linken oberen Ecke von Bild 1 decodiert der „Loader" die Befehle innerhalb der Felder, wobei er je ein Feld in jedem 50-ns-Zyklus extrahiert. Ein Feld besteht aus dem Opcode, der in der Länge von 1 Byte bis zu 3 Byte variieren kann oder einer sogenannten „Extension", wie zum Beispiel einem Displacement oder Immediate-Value. Der Loader ist verantwortlich für die Vorverarbeitung der Befehle, die Weiterleitung der Adreßinformationen zur Adresseneinheit und die Festlegung der Mikrocode-Start-Adresse für die Execution-Einheit.

Während die CPUs 32032 und 32332 über einen „Fetch" die „Pipeline" decodieren und abarbeiten, beinhaltet der Typ 32532 eine zusätzliche Pipelinestufe, die Adressenberechnungen durchführt. Es gibt darüber hinaus eine zusätzliche Pufferung, um den Datenfluß durch die Pipeline reibungsloser zu gestalten. Im Durchschnitt können zu jeder Zeit fünf Befehle durch die Pipeline übertragen werden.

Die Adresseneinheit ist verantwortlich für die Anforderung der Quellenoperanden aus dem Speicher. Um einen effizienten Zugriff auf komplexe Datenstrukturen wie Records und Arrays zu ermöglichen, bietet die Architektur der Serie 32000 verschiedene fortgeschrittene Hochsprachen-Adressierungsarten. Die Art, in der Operanden-Adressen behandelt werden, gibt ein gutes Beispiel dafür, wie Änderungen in der Implementierung substantielle Leistungsverbesserungen bringen, ohne daß erkennbare Änderungen der Architektur notwendig sind.

Beim 32032 und seinen Abkömmlingen manipuliert beispielsweise dieselbe CPU die Operanden und berechnet gleichzeitig die Operanden-Adressen. Beim 32332 kommt eine separate ALU zur Adressenberechnung zum Einsatz, um für zahlreiche Befehle die Ausführungszeit zu reduzieren. In der CPU 32532 sind Adressenberechnungen und Operanden-Zugriffe vollständig innerhalb der Pipeline integriert, was zu einem schnelleren Durchsatz führt.

Das Registerfile besitzt zwei Ports, wobei eine Port zum Lesen der Operanden für Adreßberechnungen und Befehlsausführungen dient, während das andere Port hauptsächlich als Schreibport zum Zurückschreiben von Ergebnissen in das File benutzt wird. In einigen Fällen, wie zum Beispiel bei String-Verarbeitungsbefehlen, wird das zweite Port auch zum Schreiben von Daten aus dem File benutzt.

Verschiedene interne Busse verbinden den Loader, die Adreßeinheit und die Execution-Unit mit der Speicherverwaltungs-Einheit sowie den Op-Chip-Puffern, die in den meisten Applikationen als Cache-Speicher fungieren. Die Integration der Speicherverwaltungs-Funktionen auf dem Chip ist eine naheliegende Möglichkeit, die größere Integrationsdichte auszunutzen, die

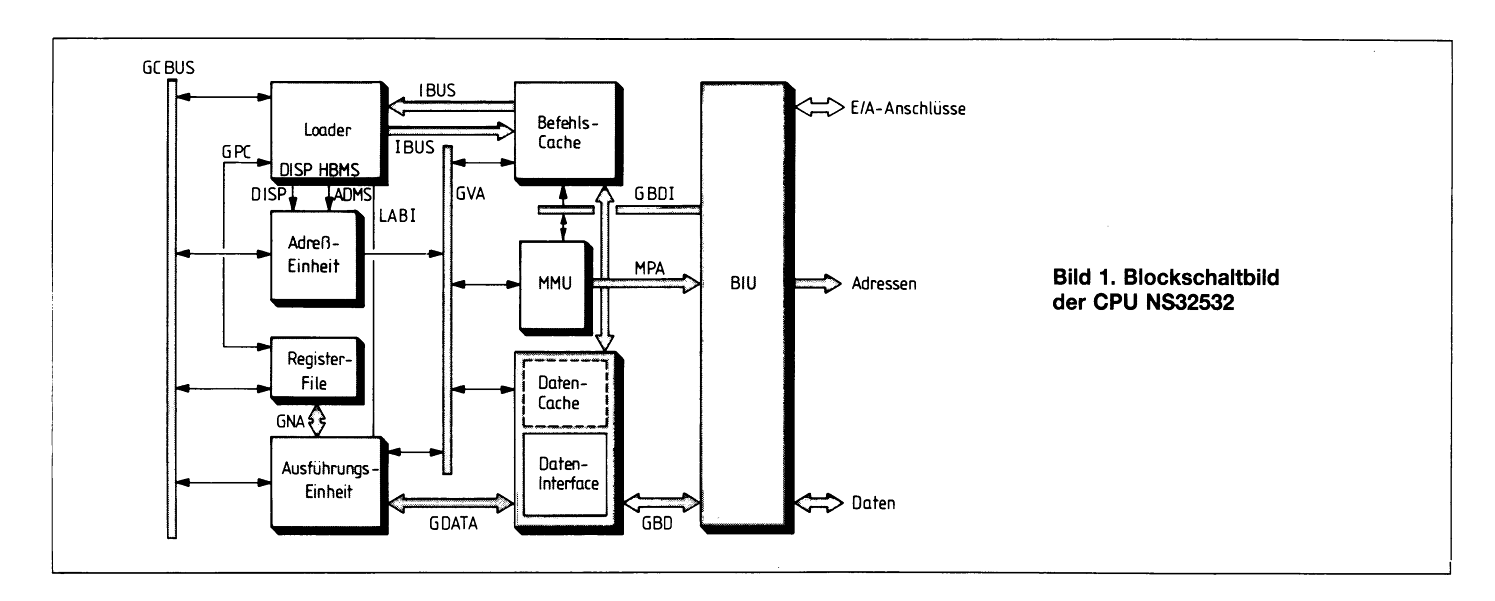


Bild 1. Blockschaltbild der CPU NS32532

durch den Einsatz eines modernen CMOS-Prozesses möglich wird. Der Wert eines On-Chip-Cache-Puffers mag auf den ersten Blick nicht deutlich werden. Es ergibt sich das Problem, daß unzureichend ausgelegter Cache kaum Vorteile bringt. Wenn er darüber hinaus noch durch eine langsame Invalidation-Prozedur ergänzt wird, so kann daraus sogar eine Leistungsreduzierung resultieren.

National Semiconductor war immer der Ansicht, daß ein On-Chip-Cache nur dann angeboten werden sollte, wenn er sowohl Daten wie Befehle verarbeiten kann und groß genug ausgelegt wird, daß eine ausrichend hohe Cache-Hit-Rate sichergestellt ist. Dies ist der Grund, warum der 32332 eine spezielle Unterstützung für externe Cache durch das verzögerte Sampling der Ready-Leitung bietet, aber keinen On-Chip-Cache beinhaltet.

Wie Bild 1 zeigt, enthält der Typ 32532 zwei separate Caches, einen für Daten und einen für Befehle. Damit wird beispielsweise die erhebliche Beschränkung in Form des nur für Befehle nutzbaren Cache überwunden, wie sie beim 68020 existiert. In der ursprünglichen Konfiguration war der Datencache für 512 Byte ausgelegt, während der Befehlscache 256 Byte speichern konnte. Die Entwickler haben jedoch die Möglichkeit offengehalten, bei zukünftigen CPUs größere Caches zu integrieren. Darüber hinaus lassen sich die Caches abschalten und die Puffer sich stattdessen als normale Speicher nutzen.

Viele Echtzeit-Applikationen beinhalten beispielsweise Algorithmen auf der Basis schneller Iterationen von kurzen Routinen. Die Abarbeitung der Basisroutine außerhalb des On-Chip-Befehlspuffers verkürzt die Ablaufzeit, weil externe Speicherzugriffe nicht nötig sind. In solchen Fällen kann die Befehlspuffer-Cache-Funktion abgeschaltet und der Pufferspeicher dazu benutzt werden, die Elementarroutine zwischenzuspeichern. Die Vorteile von On-Chip-Datacaches können sich bei Multiprozessor-Systemen mit gemeinsam genutzten Speicher entsprechend leicht ins Negative umkehren, da hier ein großer Aufwand dafür notwendig ist, sicherzustellen, daß Änderungen in irgendeinem Cache in gleicher Weise bei allen anderen Caches stattfinden. Auch aus diesem Grunde läßt sich die Datenpuffer-Cache-Funktion ausschalten.

In der Mehrheit der Applikationen werden natürlich beide Caches in Funktion sein, so daß die am meisten benutzten Befehle und Daten ohne externe Speicherreferenzen verfügbar sind. Beide Caches sind in engem Zusammenhang mit andren Funktionsblöcken integriert. So ist der Loader zum Beispiel direkt mit dem Befehlscache verbunden und leitet vorausberechnete Sprungadressen an den Cache weiter, so daß korrekt vorherbestimmte Verzweigungen nur eine Verzögerung von zwei Zyklen verursachen. Darüber hinaus übernimmt der Befehlscache per Burst-Mode-Transfer verschiedene Befehle aus dem externen Speicher, wenn der Systembus nicht durch andere Operationen belegt ist, und leitet sie an den Loader für nachfolgende Interpretationen weiter.

Die MMU, der Datencache und die Adresseneinheit sind mit einem internen virtuellen Adreßbus verbunden. Die Adresseneinheit liefert virtuelle Adressen parallel an den Datencache und die MMU, so daß die Cache-Prüfung und die Übersetzung der virtuelen in die physikalische Adresse gleichzeitig begonnen werden. Dies erspart Zeit bei der Anforderung von Daten aus dem externen Speicher, falls der Cache nicht zur Verfügung steht.

Weil eine CPU viel Zeit für die Kommunikation mit externen Speicher- und Hilfseinheiten verbraucht, kann die Schnittstelle zwischen der CPU und den anderen Teilen des Systems einen gravierenden Einfluß auf den Gesamtdurchsatz des Systems haben. Die höhere Leistungsfähigkeit des Prozessor 32532 stellt auch höhere Anforderungen an das System-Interface. Es wurden deshalb in diesem Bereich einige Änderungen vorgenommen, die *Tabelle 2* zeigt.

Die augenscheinlichste Veränderung sind die separaten 32-Bit-Busse für Adressen und Daten. Bislang machte es die Effizienz der Serie-32000-Architektur möglich, die Leistungsziele der verschiedenen Prozessoren mit einem relativ langsamen, aber sehr kostengünstigen Multiplex-Bus zu erreichen. Die sehr hohen Leistungsvorgaben beim 32532 erforderten jedoch ein Abgehen von der Multiplex-Busstruktur.

Der Einsatz von separaten Adreß- und Datenbussen in Verbindung mit der Realisierung interner Speicherverwaltungs-Funktionen brachte eine erhebliche Vergrößerung der verfügbaren Busbandbreite. Die CPUs 32032 und 32016 benötigten beim Betrieb mit einer MMU 32082 fünf Zyklen für die Bustransaktionen. Dies wurde bei der CPU 32332 auf vier Zyklen reduziert. Die CPU 32532 bietet noch schnellere Bustransaktionen: hier sind nur zwei Taktzyklen erforderlich.

Dieselbe Verbesserung wurde auch auf das 32332-Burst-Protokoll angewandt, so daß der Burst-Transfer nun innerhalb eines Zyklus stattfindet, anstatt wie bisher zwei zu benötigen. Ein wesentlicher Vorzug des Burst-Modes ist die Verringerung der internen Probleme zwischen Datentransfers und Operanden-Zugriffen. Die Fähigkeit zum Pre-Fetch von Befehlen unter Verwendung des sehr schnellen Burst-Mode-Transfers verbessert die Effizienz des Befehls-Cache um durchschnittlich 70 Prozent, verglichen mit einer Steigerung von etwa 50 Prozent beim Befehlscache des 68020-Prozes-

Auch die Bus-Wiederholfunktion konnte im Vergleich zu der des 32332 weiter verzögert werden, was speziell in solchen Systemen hilfreich ist, wo auch große extern implementierte Caches erforderlich sind.

Tabelle 2. Änderungen des System-Interface

- Separate 32-Bit-Adreß- und Datenbusse
- Schnellere Bustransaktionen
- Burst-Transfers in einem Taktzyklus
- Verzögerte Buswiederholung

Daten- und Befehlsfluß durch die Pipeline

Um den Fluß von Daten und Befehlen durch die Pipeline zu illustrieren, zeigt *Bild 2* den Verlauf einer typischen 32-Bit-ADD-Instruktion, in diesem Falle ADDD 4 (FP), RO. Beim Start dieser Sequenz wird vorausgesetzt, daß die Information über die Adressierungsart bereits für die Adreßeinheit verfügbar ist. Dies wird normalerweise der Fall sein, weil der Loader schnell genug ist, die Adreß- und Execution-Einheiten mit den zugehörigen Befehlsinformationen zu versorgen. Der GC-Bus, der die Adreßeinheit und die Fileregister verbindet, ist „vorgeladen“, was bedeutet, daß er eine spezielle Steuereinheit beinhaltet, die in der Lage ist, die angesprochenen Register auszuwählen und deren Inhalt während des ersten 50-ns-Taktzyklus auszulesen. Am Ende des ersten Taktzyklus wurde der Inhalt des Base-Register (in diesem Falle der Frame-Pointer) vom Registerfile über den GC-Bus gelesen.

Während des nächsten Zyklus wird die Verschiebung, die in einem Befehl festgelegt ist, zu der Bezugsadresse addiert, um die virtuelle Adresse des Operanden zu ergeben. Gleichzeitig erfolgt die Anforderung des Inahlts von R0 über den GC-Bus. Nach zwei Zyklen hat die Execution-Einheit deshalb die Mikrocode-Anfangsadresse vom Loader erhalten, und die Operanden-Adresse wurde berechnet. Die Prozedur kann nun für den nächsten Befehl wiederholt werden, der bereits durch den Loader vorverarbeitet wurde, während die MMU und die ALU mit der Verarbeitung der vorherigen Instruktion fortfahren.

Die effektive Adresse, die berechnet wurde, ist eine virtuelle Adresse, die an die MMU weitergeleitet wird. Falls die Daten bereits im RAM resident sind, was bei rund 98 % aller Speicherzugriffe der Fall ist, so steht die physikalische Adresse in der zweiten Hälfte des dritten Zyklus zur Verfügung. Am Beginn des vierten Zyklus erscheint die physikalische Adresse am Adreßanschluß, vorausgesetzt, der Bus ist frei. Außerdem liegt sie gleichzeitig am Datencache an. In diesem Stadium wird deshalb die erste Phase einer Bustransaktion gestartet, angezeigt durch T1 in Bild 2.

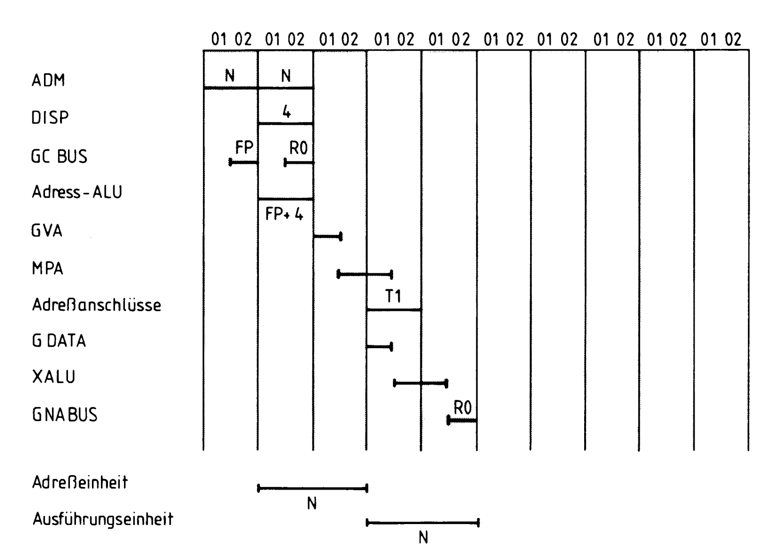


Bild 2. Das Pipeline-Flußdiagramm

Was nun weiter geschieht, hängt davon ab, ob die Daten bereits im Cache sind. Ist dies der Fall, so sind sie unmittelbar auf dem internen GDATA-Bus verfügbar, und der externe Speicherverweis wird annulliert. Die Daten werden nun an die Execution-ALU gesendet, die Addition durchgeführt und das Ergebnis in R0 zurückgespeichert, wie Bild 2 zeigt. Zum Zwecke der Abschätzung des Systemdurchsatzes kann die Sequenz vom Start des zweiten Zyklus an betrachtet werden. Das GC-Precharge, im ersten Zyklus zu sehen, überlappt normalerweise die Adreßberechnung der vorherigen Instruktion. Es gibt deshalb zwei Zyklen in der Adreßeinheit und zwei in der Execution-Einheit.

Dies bedeutet in bezug auf den Systemdurchsatz, daß die am häufigsten benutzten Instruktionen wie Register-zu-Register-, Register-zu-Speicher- und Speicher-zu-Register-Operationen eine Basis-Ausführungszeit von zwei Zyklen haben. Bei einer Taktfrequenz von 20 MHz ergibt sich ein Spitzendurchsatz von 10 MIPS. In der Praxis ist dieser Wert natürlich niedriger, weil zwei Adreßbefehle benutzt werden und es vorkommen kann, daß der Cache nicht zur Verfügung steht.

Bild 3 zeigt auch den Pipeline-Fluß, wenn ein Ausfall des Datencache auftritt. In diesem Falle wird die externe Bezugsadresse nicht annulliert und es ergibt sich eine Verzögerung von zwei Zyklen, bevor die Daten für die Execution-ALU verfügbar sind. Die Auswirkung eines Cache-Ausfalls besteht deshalb darin, daß die Befehlsausführungszeit sich auf 100 ns vergrößert.

Verzögerungen können darüber hinaus durch zeitweilige Engpässe in der Pipeline entstehen. Dies ist zum Beispiel der Fall, wenn R0 i einer Instruktion als Zielregister und in der folgenden Instruktion als Basisregister benutzt wird, so wird der reibungslose Fluß innerhalb der Pipeline unterbrochen, weil zwei interne Ressourcen zur gleichen Zeit Zugriff auf R0 fordern. Die Art, wie Instruktionen auftreten, kann deshalb die Ausführungsgeschwindigkeit beeinflussen. In der Zukunft verfügbare Compiler werden in der Lage sein, diesen Umstand auszunutzen.

Echtzeit-Unterstützung

Obwohl die meisten Mikroprozessor-Architekturen für allgemeine Computeranwendungen optimiert sind, beinhaltet der Typ 32532 verschiedene Neuerungen, die speziell auf Echtzeit-Applikationen ausgerichtet sind. Weil die Geschwindigkeit im allgemeinen ein wichtiges Kriterium bei Echtzeit-Applikationen ist, sind die Möglichkeiten des 32532 zur Unterstützung von Echtzeit-Anwendungen alle im Hinblick auf die Steigerung der Geschwindigkeit entwickelt worden.

Tabelle 3 zeigt die Verbesserungen, die in bezug auf die Interrupt-Antwortzeit, die Context-Switching-Time und die Datenspeicher-Bandbreite im Vergleich mit den CPUs 32032 und 32332 erreicht wurden. Die wesentli-

chen Geschwindigkeits-Verbesserungen bei der CPU 32532 sind ebenso das Resultat einiger neuer Möglichkeiten wie das des On-Chip-Caches und der höheren Taktfrequenz. Eines der neuen Merkmale, als „Direct-Exception-Mode" bezeichnet, ermöglicht dem Anwender zwischen Adressierungs-Flexibilität und Interrupt-Antwortzeit zu wählen. Den normalen Ablauf beim Aufruf eines Interrupts oder einer Trap-Service-Routine zeigt *Bild 3*. Nachdem die Rücksprung-Adresse und das PSR/MOD-Registerpaar auf dem Interrupt-Stack gesichert sind, wird der Interrupt-Vektor als Index in der Interrupt-Dispatch-Tabelle verwendet. Die Basisadresse steht im INTBASE-Register der CPU. Der Index zeigt auf einen externen 32-Bit-Prozedur-Deskriptor, der dann in einem externen Prozedur-Aufruf benutzt wird.

Der externe Prozedur-Deskriptor besteht aus zwei 16-Bit-Worten, von denen eines den neuen MOD-Registerwert annimmt. Unter Beutzung des neuen MOD-Registerwertes als Zeiger in der Modultabelle entstehen ein neuer Static-Base-Pointer, ein neuer Link-Base-Pointer und ein neuer Programm-Base-Pointer. Die verbliebene Hälfte des externen Prozedur-Deskriptors wird nun dem neuen Programm-Basis-Zeiger hinzugefügt, um die Anfangsadresse für die Service-Routine zu erhalten.

Während dieses Konzept des Exception-Handlings für die meisten Applikationen ideal geeignet ist, kann die Zeit, die zum Ausführen der Sequenz notwendig ist, diese Vorteile in manchen Echtzeit-Applikationen aufwiegen, weil es hier auf die Geschwindigkeit der Antwort auf ein externes Signal ankommt. In solchen Fällen ist speziell der neue Direkt-Exception-Modus hilfreich. In diesem Modus, mit den man durch das Setzen eines Bits in den erweiterten Programm-Status-Register

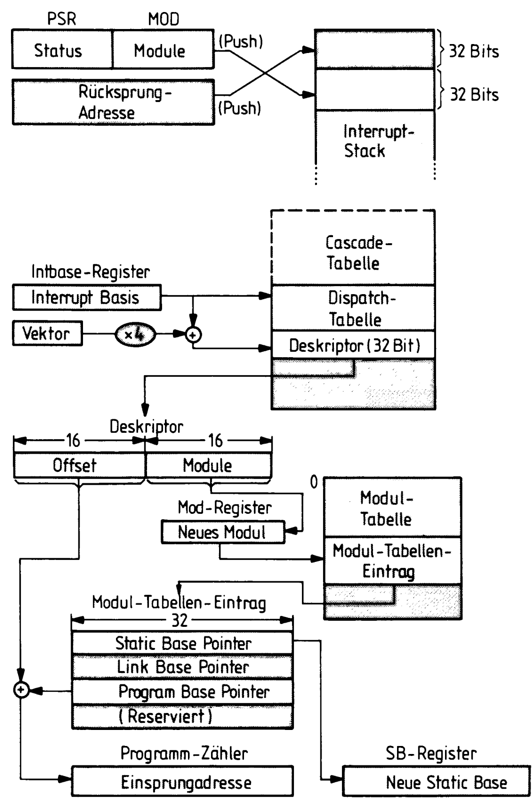


Bild 3. Aufruf-Sequenz für die Exception-Service-Routine

Tabelle 3. Verbesserungen der Echtzeit-Eigenschaften

	32032 10 MHz	32332 15 MHz	32532 20 MHz
Vektor-Interrupt-Antwortzeit	8,5 µs	3,8 µs	1,5 µs
Context-Swap	59,5 µs	19,1 µs	3,8 µs (100 % Hits) 7,5 µs (No Hits)
Datenspeicher-Transfer-Geschwindigkeit	8 MB/s	15 MB/s	80 MB/s (On-Chip) 40 MB/s (Off-Chip)

gelangt, wird die Anfangsadresse in der Interrupt-Dispatch-Tabelle als neuer Programm-Zählwert interpretiert und nicht als ein externer Prozedur-Deskriptor.

Der Nachteil dieses Konzeptes ist natürlich, daß die existierende Execution-Umgebung für die Interrupt-Service-Routine verwendet wird. Der Static-Base-Pointer wird beispielsweise nicht verändert, so daß auf alle statischen Daten, die von der Service-Routine angefordert werden, über eine absolute Adresse zugegriffen werden muß. Jedoch reduzieren die bei Einsatz dieses Modus „eingesparten" zehn Zyklen die Antwortzeit um 500 ns.

Ein anderes Merkmal zur Beschleunigung des Interrupt-Service ist der direkte Zugriff auf den User-Stackpointer. Dies wird durch zwei neue Befehle ermöglicht, die es erlauben, den USP im Supervisor-Modus zu laden oder zu speichern, ohne das S-Bit im Programm-Status-Register zu ändern.

Der Direkt-Exception-Modus ist eine von mehreren Verbesserungen der Exception-Verarbeitung. Darüber hinaus gibt es auch eine neue maskierbare Integer-Overflow-Trap, die speziell für die Exception-Prüfung in ADA-Compilern und ähnlichen Applikationen hilfreich ist. Page-Crossing-Beschränkungen während Befehlswiederholungen konnten beseitigt werden. Die MMU beinhaltet nun auch einige zusätzliche Debug-Möglichkeiten, speziell in bezug auf Pipeline-Operationen.

Natürlich haben die Leistungsverbesserungen beim Prozessor 32532 einige Änderungen in der Architektur des Bausteins notwendig gemacht. Es wurden jedoch alle Anstrengungen unternommen, diese notwendigen Änderungen in Einklang mit dem Prinzip der Familien-Kompatibilität zu bringen. Die Änderungen in der Architektur beschränken sich auf das absolute Minimum und betreffen nur solche neuen Funktionen, die auf die bisherigen CPUs nicht anwendbar sind.

Die neuen Befehle sind zum Beispiel: Load/Store-USP und -CFG, Load/Store-Debug-Registers und Cache-Invalidate. Dies sind Erweiterungen zum Priveleged-Mode-Instruction-Set. Der User-Mode-Instruction-Set ist unverändert. Obwohl es aus diesem Grunde nicht mehr länger zutrifft, daß dieselbe Software auf allen CPUs der 32000-Serie ohne Änderungen läuft, sind Software-Modifikationen, die notwendig sind, um die volle Leistungsfähigkeit des 32532 auszuschöpfen, im allgemeinen auf das Betriebssystem beschränkt und weniger auf die Applikationsprogramme.

Sonderheft Nr. 218
Preis DM 28.–
öS 210.–, sfr 28.–

Elektronik

Das 32000-Sonderheft

Bausteine, Programmierung Systeme, Anwendungen

```
/* open the file */
fildes = open(filename, 0)) < 0
printf("Cannot find %s
exit(1);
```

```
/* the buffer for writing */
char buffer[512];
/* the filename */
char *filename = "a_large_file";
/* a counter counting blocks read
register int i;
/* the file descriptor
int fildes;
```

```
#endif
#ifdef ASSIGN
/* The second way of doing things — wi
{
```

```
#define BLOCKS
/* the buffer
char buffer[512];
/* file descrip
int fid[2];

main()
{
```

```
/* Eratosthenes
#define TRUE
#define FALSE
#define SIZ
```

```
/* seek to it */
```